TURING 图灵程序设计丛书



深入HTML5应用开发

HTML5 Geolocation

Making Isometric Social Real-Time Games with HTML5, CSS3, and JavaScript

[美] *Anthony T. Holdener III*
[阿根廷] *Mario Andrés Pagella* 著
秦绪文 李松峰 译

O'REILLY®

人民邮电出版社
POSTS & TELECOM PRESS

数字版权声明

 图灵程序设计丛书

深入HTML5应用开发

HTML5 Geolocation
Making Isometric Social Real-Time Games with
HTML5, CSS3, and JavaScript

[美] Anthony T. Holdener III
[阿根廷] Mario Andrés Pagella 著
秦绪文 李松峰 译

O'REILLY®
Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo
O'Reilly Media, Inc.授权人民邮电出版社出版

人民邮电出版社
北 京

图书在版编目（C I P）数据

深入HTML 5应用开发 / (美) 赫尔德尔
(Holdener,A.T.) , (阿根廷) 佩奇拉 (Pagella,M.A.)
著 ; 秦绪文, 李松峰译. -- 北京 : 人民邮电出版社,
2012.3
（图灵程序设计丛书）
ISBN 978-7-115-27494-6

Ⅰ. ①深… Ⅱ. ①赫… ②佩… ③秦… ④李… Ⅲ.
①超文本标记语言，HTML 5－程序设计 Ⅳ. ①TP312

中国版本图书馆CIP数据核字(2012)第023548号

内 容 提 要

本书分两部分，分别对应O'Reilly公司出版的*HTML5 Geolocation*和*Making Isometric Social Real-Time Games with HTML5, CSS3, and JavaScript*。第一部分介绍的是W3C Geolocation API，共6章。第二部分介绍的是使用HTML5、CSS3和JavaScript，利用等轴投影原理开发一款融入社交元素的实时游戏，共5章。

本书适合所有使用HTML5开发Web应用的人员阅读。

图灵程序设计丛书

深入HTML5应用开发

◆ 著　[美] Anthony T. Holdener III
　　[阿根廷] Mario Andrés Pagella
译　秦绪文　李松峰
责任编辑　朱　巍
执行编辑　丁晓昀　刘美英

◆ 人民邮电出版社出版发行　北京市崇文区夕照寺街14号
邮编　100061　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京　印刷

◆ 开本：800×1000　1/16
印张：16.75
字数：286千字　2012年3月第1版
印数：1－4 000册　2012年3月北京第1次印刷
著作权合同登记号　图字：01-2012-0970号
ISBN 978-7-115-27494-6

定价：59.00元

读者服务热线：(010)51095186转604　印装质量热线：(010)67129223
反盗版热线：(010)67171154

版权声明

O’Reilly Media, Inc.介绍

O’Reilly Media 通过图书、杂志、在线服务、调查研究和会议等方式传播创新知识。自 1978 年开始，O’Reilly 一直都是前沿发展的见证者和推动者。超级极客们正在开创着未来，而我们关注真正重要的技术趋势——通过放大那些“细微的信号”来刺激社会对新科技的应用。作为技术社区中活跃的参与者，O’Reilly 的发展充满了对创新的倡导、创造和发扬光大。

O’Reilly 为软件开发人员带来革命性的“动物书”；创建第一个商业网站（GNN）；组织了影响深远的开放源代码峰会，以至于开源软件运动以此命名；创立了 Make 杂志，从而成为 DIY 革命的主要先锋；公司一如既往地通过多种形式缔结信息与人的纽带。O’Reilly 的会议和峰会聚集了众多超级极客和高瞻远瞩的商业领袖，共同描绘出开创新产业的革命性思想。作为技术人士获取信息的选择，O’Reilly 现在还将先锋专家的知识传递给普通的计算机用户。无论是通过书籍出版、在线服务或者面授课程，每一项 O’Reilly 的产品都反映了公司不可动摇的理念——信息是激发创新的力量。

业界评论

“O’Reilly Radar 博客有口皆碑。”

——Wired

“O’Reilly 凭借一系列（真希望当初我也想到了）非凡想法建立了数百万美元的业务。”

——Business 2.0

“O’Reilly Conference 是聚集关键思想领袖的绝对典范。”

——CRN

“一本 O’Reilly 的书就代表一个有用、有前途、需要学习的主题。”

——Irish Times

“Tim 是位特立独行的商人，他不光放眼于最长远、最广阔的视野并且切实地按照 Yogi Berra 的建议去做了：‘如果你在路上遇到岔路口，走小路（岔路）。’回顾过去，Tim 似乎每一次都选择了小路，而且有几次都是一闪即逝的机会，尽管大路也不错。”

——Linux Journal

目录

HTML5 Geolocation

前言……3

第 1 章　路在何方……9

1.1　地理定位的历史……11
1.1.1　公元前的地理定位……11
1.1.2　探险技术……11
1.1.3　20 世纪的定位……13
1.2　GPS 的民用化……14
1.3　今天的地理定位……15
1.4　基本知识……15
1.5　定位方法……16
1.5.1　GPS……17
1.5.2　IP 地址……18
1.5.3　GSM/CDMA Cell ID……19
1.5.4　Wi-Fi 和蓝牙的 MAC 地址……20
1.6　位置与基于位置的服务……20
1.7　今天的地理定位……21
1.7.1　手机应用……21
1.7.2　增强现实应用……24

第 2 章　地理定位：不仅是经纬度……27

2.1　坐标系统……29
2.2　大地测量系统与基准点……32

2.2.1 地球的形状 32
2.2.2 公共基准 33
2.2.3 地图投影 35
2.3 高度、路线与速度 35
2.3.1 大地高度 36
2.3.2 路线 37
2.3.3 速度 37
2.4 精确度 38

第 3 章 地理定位 API 编程 39

3.1 W3C Geolocation API 41
3.1.1 当前的 API 支持情况 41
3.1.2 其他浏览器的解决方案 42
3.2 功能更完备的 W3C Geolocation API 44
3.3 Geolocation 对象 45
3.4 取得用户位置 46
3.4.1 PositionOptions 46
3.4.2 缓存的位置信息 47
3.5 更新用户位置 48
3.5.1 不需要轮询 49
3.5.2 清除监视操作 49
3.6 处理成功的请求 50
3.6.1 Position 对象 50
3.6.2 Coordinates 对象 50
3.7 处理请求返回的错误 53
3.8 隐私问题 55

第 4 章 地理定位和地图 API 57

4.1 Google 地图示例 59
4.1.1 Google Maps API 简介 59
4.1.2 向 Google 地图中添加地理定位 63
4.2 ArcGIS JavaScript API 的例子 70
4.2.1 ArcGIS JavaScript API 简介 70
4.2.2 向 Esri 地图中添加地理定位 73

第 5 章 保存地理信息 81

5.1 KML 83
5.2 Shapefile 86
5.3 数据库 89
5.3.1 SDE 89

5.3.2 PostGIS ······ 90
5.3.3 MySQL ······ 90

第 6 章 基于地理定位开发应用 ······ 93

6.1 地理营销 ······ 96
6.1.1 特价与新品 ······ 96
6.1.2 众包 ······ 96
6.1.3 特殊化 ······ 97
6.2 地理社交 ······ 98
6.2.1 持续增长 ······ 98
6.2.2 自动签到 ······ 99
6.2.3 双向数据 ······ 99
6.3 地理标签 ······ 100
6.3.1 数字媒体与地理标签 ······ 100
6.3.2 隐私与地理标签 ······ 100
6.4 地理应用 ······ 101
6.4.1 安全 / 跟踪 ······ 101
6.4.2 打车 ······ 101
6.4.3 搜索 ······ 102
6.4.4 移动商务 ······ 102
6.4.5 其他应用 ······ 102
6.5 HTML5 与地理定位 ······ 102
6.5.1 辅助 LBS 的 Web 应用 ······ 103
6.5.2 基于 Web 的 LBS ······ 110

HTML5：等轴实时游戏开发

前言 ······ 113

第 1 章 图形基础：画布与精灵 ······ 119

1.1 使用 canvas 对象 ······ 121
1.2 创建平滑的动画 ······ 129
1.3 使用精灵 ······ 133
1.4 操作像素 ······ 138
1.5 为图像选择渲染方法 ······ 149

第 2 章 理解等轴游戏 ······ 163

第 3 章　游戏界面设计 181

3.1　Web 游戏中的 GUI 设计和交互 183
3.2　实现 GUI 185

第 4 章　HTML5 声音及处理优化 201

4.1　通过 `audio` 元素添加声音 203
4.2　用 Web Workers API 执行大计算量任务 213
4.3　本地存储和会话存储 221

第 5 章　推向市场 227

5.1　预防作弊及服务器端工作 229
5.2　通往最终游戏的路 235
5.3　对游戏作最后修饰 248
5.4　使用 Facebook 添加社交功能 254

HTML5 Geolocation

[美] Anthony T. Holdener III 著



前言

本书第一部分详细介绍 W3C Geolocation API[1]。Geolocation API 支持以脚本方式访问与主机设备相关的地理位置信息。这个 API 定义了一组对象，在 JavaScript 中使用这些对象可以探知运行代码的设备位置。



术语地理定位（geolocation）有时指找到某人位置的动作，有时候也可以指实际的位置。

W3C Geolocation API 为浏览器带来了令人难以置信的能力。此前，只有那些在特殊设备上编写本地地理定位应用的开发人员才有可能使用定位服务。现在，开发人员可以自由地直接在浏览器中针对 Web 编写地理定位应用，而且这些应用还具备“一次编写，随处部署”的优点。

关于这部分的标题

在继续讲下去之前，我想需要为这部分的标题 HTML5 Geolocation 说声抱歉！从技术角度讲，Geolocation API 并不是 W3C HTML5 规范的一部分。因此，HTML5 Geolocation API 这种提法的确是错误的，我知道这一点。

话虽这样说，我还是想请大家用 Google 搜一搜“Geolocation API”或者“HTML5 APIs”，看看结果中有多少条目以“HTML5 Geolocation”作为标题。相信你会发现，这样的条目非常少，除了 W3C Geolocation API Working Draft——这个标题省略了 HTML5 字样。此外，我在加州棕榈泉市（Palm Springs）举办的 2011

注 1：Geolocation API Specification: W3C Candidate Recommendation 07 September 2010。编辑 Andrei Popescu，Google 股份有限公司，http://www.w3.org/TR/geolocation-API/。

Esri Developer Summit 上，听了很多场有关 JavaScript 的演讲。每一位演讲人在提到 Geolocation API 时，也会提到 HTML5，无一例外。这些人熟悉自己的 GIS（Geographic Information System，地理信息系统），以 GIS 为生，而且都在世界领先的 GIS 软件公司工作。

事实很简单，说到 Geolocation，大家都会联想到 HTML5。为了避免不使用 HTML5 可能给读者带来的疑惑，加之我和我的编辑都谁想不出来给这部分起个什么名字才更时髦，结果就只能选 HTML5 Geolocation 了。

读者对象

这部分适合所有想在 Web 应用中使用 W3C Geolocation API 的开发人员阅读。前几章详细介绍什么是地理定位，地理定位的历史以及地理定位今天的应用现状。

前几章是有关地理定位的简要教程，可以帮助读者理解这个 API 的大致情况。假如你有 GIS 行业的从业经验，只想知道怎么在自己的应用中使用这个新的 API，或者你已经对地理定位了如指掌，可以直接看第 3 章关于 API 实际应用的内容。

开发人员肯定会对第 3 章和第 4 章特别感兴趣，因为这两章基于代码和示例讨论了这个 API 的使用方法。非程序员也大致能够看懂这两章的内容，从而对这个 API 能够做什么有一个更深入的理解。第 6 章探讨了地理定位对我们未来生活的影响，介绍了使用 Geolocation API 开发实际的应用。

排版规范

本书使用的排版规范如下所示。

- 楷体

用于表示新的术语、URL、电子邮件地址、集合名、数据库名、文件名及文件扩展名。

- 等宽字体

表示程序片段，也在段落中表示程序中使用的变量、函数名、命令行实用工具、环境变量、语句和关键词等元素。

- 等宽加粗

这种字体表示用户需要手动输入的命令或者相应的文本。

- 斜体

用户需要根据自己所提供的值或由上下文所确定的值进行更改的部分。



这个图标代表小窍门、建议或者注意。

使用示例代码

让我们助你一臂之力。也许你要在自己的程序或文档中用到本书中的代码。除非大段大段地使用，否则不必与我们联系取得授权。例如，无需请求许可，就可以用本书中的几段代码写成一个程序。但是销售或者发布 O'Reilly 图书中代码的光盘则必须事先获得授权。引用书中的代码来回答问题也无需授权。将大段的示例代码整合到你自己的产品文档中则必须经过许可。

我们非常希望你能标明出处，但并不强求。出处一般含有书名、作者、出版商和 ISBN，例如"*HTML5 Geolocation*，Anthony T. Holdener Ⅲ（O'Reilly，2011）版权所有，978-1-449-30472-0"。

如果有关于使用代码的未尽事宜，可以随时与我们取得联系，permissions@oreilly.com

Safari®在线图书



Safari 在线图书是应需而变的数字图书馆。它能够让你非常轻松地搜索 7500 多种技术性和创新性参考书以及视频，以便快速地找到需要的答案。

订阅后就可以访问在线图书馆内的所有页面和视频。可以在手机或其他移动设备上阅读，还能在新书上市之前抢先阅读，也能够看到还在创作中的书稿并向作者反馈意见。复制粘贴代码示例、放入收藏夹、下载部分章节、标记关键点、做笔记甚至打印页面等有用的功能可以节省大量时间。

这本书也在其中。欲访问本书英文版的电子版，或者由 O'Reilly 或其他出版社出版的相关图书，请到 http://my.safaribooksonline.com 免费注册。

我们的联系方式

请把对本书的评论和问题发给出版社。

美国：

O’Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472

中国：
北京市西城区西直门南大街 2 号成铭大厦 C 座 807 室（100035）
奥莱利技术咨询（北京）有限公司

O’Reilly 的每一本书都有专属网页，你可以在那儿找到关于本书的相关信息，包括勘误表、示例代码以及其他信息。本书的网站地址是：

http://www.oreilly.com/catalog/9781449304720/

中文书：

http://www.oreilly.com.cn/index.php?func=book&isbn=9787121140468

对于本书的评论和技术性的问题，请发送电子邮件到：

bookquestions@oreilly.com

关于本书的更多信息、会议、资源中心和网络，请访问以下网站：

http://www.oreilly.com
http://www.oreilly.com.cn

致谢

首先，特别感谢我的妻子 Sarah。在我忙于写作期间，她不仅照料一切（特别是照顾几个孩子），而且还充当了编辑的角色，用红笔给第一稿做了不少加工润色。以她优秀的写作功底，肯定提升了文章的可读性。非常高兴我能够写出一些能让你愿意看的东西来！

我想要感谢那些给我建议、评论，纠正书中错误的审稿人。你们的工作让本书更加完善，我打心眼儿里感激你们。Brian Dunn，还有 John Jackson——谢谢！

非常感谢我的编辑 Simon St. Laurent，他总是给我机会让我能写一些自己关心的主题，而且他是一位非常出色的编辑，平易近人。谢谢你对我的信任，让我再次为 O’Reilly Media 奋笔疾书。

最后，我要感谢作出贡献的其他所有人。感谢 O'Reilly Production Services 校对付出的心血，感谢 Adam Zaremba 在最后几分钟提出的修改建议。感谢 Karen 绘制了封面动物，感谢 David 把书稿排成了它应有的版式。还有，感谢 Robert 把我手绘的图形转换成了漂亮的插图。

我很荣幸能够写一篇针对 Web 的地理定位方面教程，整个写作的过程非常愉快，偶尔也会遇到点儿困难。但愿，你们喜欢！

第 1 章

路在何方



地球人，无论是谁，只要他从一个地方旅行到另一个地方，那他就使用过某种方法——尽管方式不同精度不同——来计算某个时间点自己所在的大致位置。随着技术的进步，精确地检测我们所在位置的能力也不断增强。*地理定位*这个术语恰当地描述了确定一个人、一个地点或一个东西所在地理位置的概念。在我们这个时代，地理定位需要用到能够连接互联网的设备（计算机、路由器、平板电脑等）、智能手机或者基于 GPS 的系统。

1.1 地理定位的历史

尽管在今天看来，使用内置 GPS 的设备确定我们的位置既容易也方便，但过去却不是这样。几千年来，人们发明了很多独出心裁的方法计算自己所在位置，其核心就是地理定位——使用既有技术确定实际地理位置。

1.1.1 公元前的地理定位

几千年前，人们依靠视觉上的地理定位来辅助确定自己在某一地区所处的方位。有史料记载的最早的一种视觉定位形式就是*烟雾信号*。资料显示，古代中国人、古希腊人和古印第安人都曾使用烟雾信号来导航，或者将信息传递到相当远的地方（肉眼可以看得到烟雾的地方）。烟雾信号可以反映出地形地貌，从而为领航员提供比较可靠的参考依据，同时也能够借此粗略估算出距信号地的距离。这些标志物可以帮助狩猎队或探险队找到回程。

随着文明的进步，人类对数学和自然的认识也越来越深入。古代的航海家发现了日月星辰与地球的相对位置，通过计算知道了如何利用某些“固定的”星星（如北极星）的角度来确定自己的地理位置。希腊、腓尼基、挪威、波斯以及中国等古代文明，都利用天体辅助航海、制作工具（稍后会介绍其中一些），从而能够到陆地之外去探险。能够到远离大陆很远的地方探险，最终发现新大陆，同时也将他们文明的火种远播四方。

1.1.2 探险技术

到了中世纪，海上导航主要服务于世界各地的贸易活动。15 世纪初，随着探险活动的兴起，对导航的需求也越来越大。在毫无依凭的大海上，探险队依靠更先进的工具确定自己船只的位置，才能实现跨越广袤无垠的水域进行远距离航行。

阿拉伯帝国在中世纪早期曾雄居经济强国长达 600 年之久，也为导航技术的进步作出了巨大贡献。当时的人们不仅通过河流贸易线旅行，还开辟了海上航线。阿拉伯人使用的主要导航工具是*磁罗盘*和一种名叫卡马尔（kamal）的仪器，如图 1-1 所

示。卡马尔是一种辅助确定纬度的导航设备，最早由阿拉伯船员使用，后来也流传到了印度和中国。卡马尔由一块中间带孔的木板和一条以相等距离打好结的绳子组成，绳子穿过木板中心的孔与木板相连[1]。在测量纬度时，把木板沿着绳子滑动，直到其角度恰好对着一颗恒星（比如北极星），然后根据从绳子末端到木板的绳结数就可以计算出纬度来。

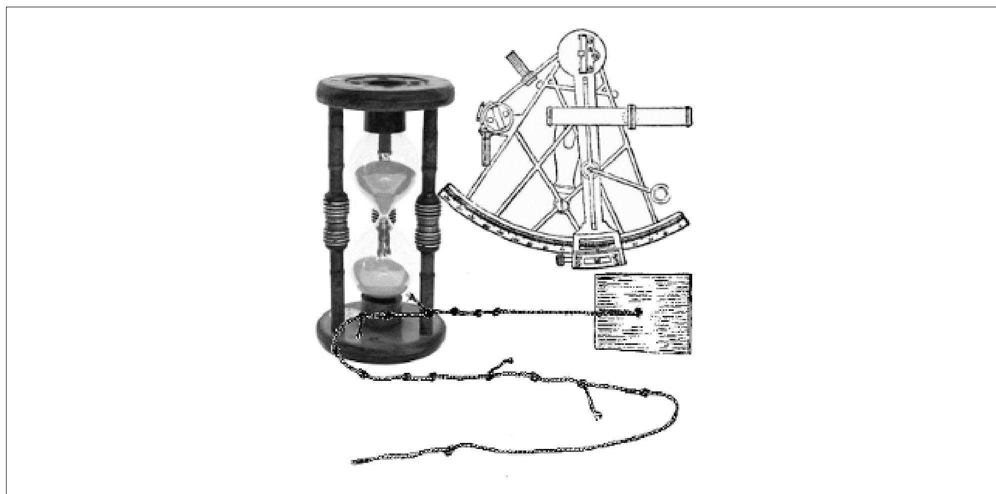


图 1-1　探险时代使用的定位工具

在此期间，船上使用的磁罗盘与今天的指南针基于相同的原理。磁针围绕一个定点旋转，停下来的时候就会与地球的磁场对齐。航海者使用指南针和卡马尔，可以知道航向并在接近赤道的海域计算出船只所在的大致纬度。

几个世纪之后，欧洲的航海家开始到更远的大海去探险。他们也引进了阿拉伯人的导航工具，但发现有的工具——特别是卡马尔，在欧洲人航海的高纬度地区难以使用。要计算纬度，他们需要一种更复杂的仪器，能够计算出太阳与星辰的夹角。最早发明的设备叫直角器（cross-staff），当时也称为雅各布连杆（Jacob's staff）。直角器与卡马尔的基本原理相同，但是多出了两个稍长一些的木杆，构成了直角。直角器最终被星盘和象限仪等更精确的仪器所取代。

星盘（astrolabe）是一个标有刻度的圆盘，用于测量太阳或某个恒星的垂直倾角。这种星盘是专门为航海制造的，能够适应大海和风浪等恶劣条件。差不多与此同时，航海家也开始使用象限仪作为对星盘的辅助。象限仪也用来度量角度，但它度量的是太阳等发光体的投射角。象限仪最初只有一个简单的支架和相应的弧形外框，但

注 1：参见 Donald Launer 著的 *Navigation Through the Ages*（Sheridan House，美国第 1 版，2009 年）。

随着时间推移，慢慢地发展成了由多个支架和弧形框组成的复杂仪器，比如戴维斯象限仪。

在历史上这一时刻出现的所有仪器，都旨在度量船只在大海上所处的纬度，但却没有很好的计算经度的方法。如果不能精确地计算出船只航行的时间和速度，要计算出经度来几乎是不可能的。在真正的计时器尚未诞生的情况下，航海者尝试使用了水钟和沙钟——沙漏就是这样一种计时器。在 18 世纪后期计时更加精确的手表或者精密记时表（chronometer）出现之前，沙漏一直都是人们的计时工具。

在能够准确地计算经度之后，航海者继续寻找计算船只纬度的更精确的方法。八分仪（octant）及最终的六分仪（sextant）取代了象限仪和星盘。图 1-1 中就有一个六分仪。六分仪用于度量两个可见对象之间的角度，而在船上则用于度量太阳或恒星与地平线之间的夹角。直到今天，六分仪依然是一个有效的导航工具，可以在没有 GPS 和无线电通信系统的时候使用，因为它不用电[2]。

1.1.3　20世纪的定位

20 世纪初，航船上开始使用无线电来检查船载精密记时表的准确度，同时通过它来确定方向（当然，也使用无线电来通信）。确定方向是通过一种叫做定向（DF，Direction Finding）的技术来实现的，所谓定向就是根据从某个信号发射器接收到的信号的方向计算出一个路径。当然，这个信号发射器并不局限于发射无线电的设备。只要尝试定向的对象能够接收到信号，任何无线设备都可以充当发射器。当接收设备把来自多个方向的信息组合到一起之后，就可以利用三角测量技术计算出发射器的位置。三角测量（triangulation）是使用两个或多个唯一的信号发射器来度量接收到的信号距离（径向距离或定向距离）的过程。图 1-2 演示了同时基于径向距离和定向距离进行三角测量以计算位置的方法。第一幅图是使用三个发射器的径向距离对一个设备作三角测量，第二幅图是使用两个发射器的定向距离对一个设备作三角测量。

1957 年，苏联发射了第一颗人造卫星——Sputnik，激发了卫星导航系统的设计思想。美国科学家发现能够基于多普勒效应监测人造卫星发射的无线信号。卫星发射的无线电信号的频率，会随着卫星接近监测点而升高，随着卫星远离监测点而下降。利用多普勒失真（Doppler distortion），科学家可以计算出任何时候卫星在其轨道中的精确位置。

注 2：参见 David Burch 著的 *Emergency Navigation: Find Your Position and Shape Your Course at Sea Even if Your Instruments Fail*（McGraw-Hill, 2008 年）。

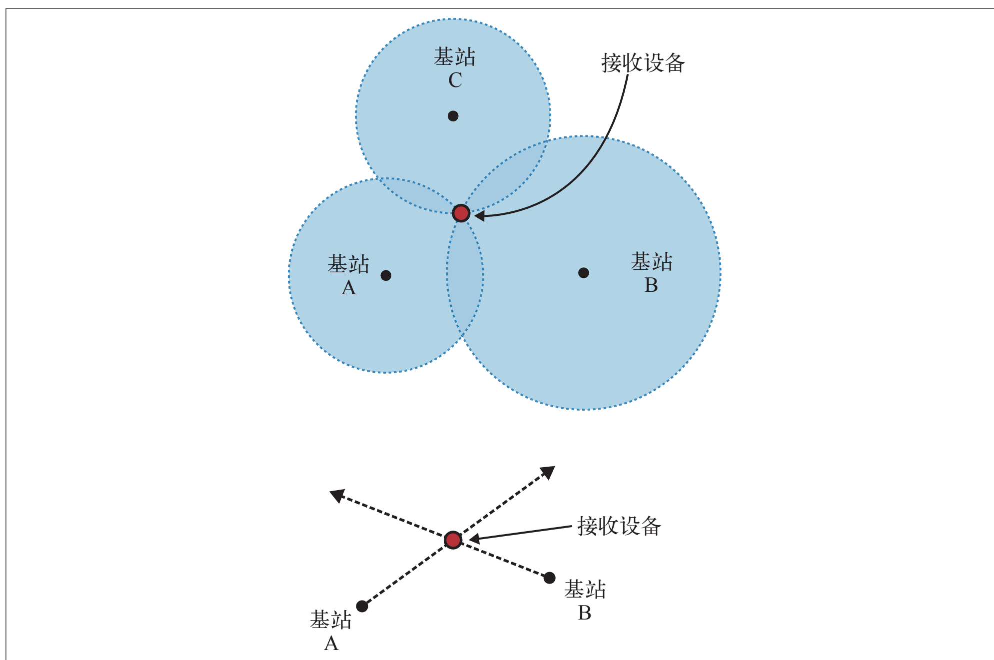


图 1-2　无线电塔的三角测量，使用了径向和定向距离的三角测量技术



多普勒效应是由奥地利物理学家克里斯琴 · 多普勒（Christian Doppler）首先发现的，该效应描述了声波在接近和远离观测者时频率的变化。救护车的汽笛声就是一个明显的例子：当救护车开近时，汽笛声会非常大（波长变短），而当救护车迅速开远时，汽笛声也会随之逐渐变得轻柔（波长变长）。如果能够度量出声调变化的速率，那么就能够估算出救护车的速度[3]。

接下来的几十年时间里，美国军方也向太空发射了很多卫星。其中，涉及导航卫星项目的包括 Transit、Timation、Project621B 和 SECOR。每个项目都吸取了上一代的经验教训，在此基础上，美国军方最终建立了 DNSS（Defense Navigation Satellite System，国防导航卫星系统）。1973 年底，DNSS 更名为 Navstar。这个 Navstar 正是我们今天熟知并常用的 GPS 系统的基础。

1.2　GPS的民用化

Navstar 最初完全是一个军用系统，民间不能使用。但 1983 年，苏联在东海击落韩国航空公司 007 班机，致使 269 名乘客和机组人员全部丧生，这一事件使情况完全

注 3：参见 Adrian Eleni 的文章“The National Center for Supercomputing Applications”(http://archive.ncsa.illinois.edu/Cyberia/Bima/doppler.html)，1995 年。

改变。在各种让人想象不到的条件下，这架飞机由于偏航飞到了苏联领空。而当时的苏联已经制定了导弹试射计划，声称该班机在执行间谍任务，于是就派空军拦截机将其击落。结果，罗纳德 · 里根总统向美国军方发布了命令，要求将其开发中的 GPS（Global Positioning System，全球定位系统）技术向民间开放，以避免类似 007 班机的事件再度发生。为了部署 GPS 阵列，从 1989 年发射第一颗卫星到 1994 年发射最后一颗卫星，总共有 24 颗卫星上天。

从第一颗卫星上天起，美国军方使用的都是质量最高的信号，而对于民用信号，则根据 SA（Selective Availability，选择可用性）原则有意进行了降级。比尔 · 克林顿总统签署了废除 SA 的命令，并于 2000 年 5 月 1 日零时生效。随着 SA 的废除，民用 GPS 的精确度从大约 100 米提高到了 20 米。

1.3 今天的地理定位

从 1978 年至今，先后有 59 颗卫星被成功地部署在了地球轨道上。到 2010 年，其中仍然有 30 颗卫星处于正常工作状态。美国计划在接下来的几年内继续向太空发射 GPS 卫星，并且已经与欧盟达成了一项合作协议，以便使用他们的伽利略卫星导航系统（预计 2014 年实施）。

如前所述，从使用烟雾信号开始，人类用来精确地确定自己所在位置的技术经历了漫长的发展过程。GPS 让精确定位成为可能，是当今及未来在地球上进行导航的系统。在了解了如何使用今天的技术确定自己的位置之后，下面我们就来看一看地理定位。

1.4 基本知识

在如图 1-3 所示的世界地图上，你的位置就是地图上的一个点。这个点涉及两方面信息：纬度和经度，GPS 软件根据它们来确定你的位置。一旦在地图上找到了准确的位置——像插了一根大头针，GPS 程序就能够为用户取得更多信息，例如附近的公司、交通状况、其他人。既然有了一个点，那应用就可以通过反向地理编码的过程来取得用户周围地区的信息。



地理编码是一种为相关文本信息（如街道地址、邮政编码）标注地理坐标的方法。反向地理编码是一个相反的过程，即基于地理坐标来使用相关的文本位置信息。



图 1-3　在地球的任何地方定位

当然，位置信息不一定来自 GPS 系统。使用什么信息、如何处理和解析位置信息，都取决于使用的设备。

1.5　定位方法

今天的计算设备可以通过多种方法获得位置信息，并不是所有设备都依赖于 GPS 卫星。以下是几种处理位置信息的方法：

- GPS
- IP 地址
- GSM/CDMA Cell ID
- Wi-Fi 和蓝牙的 MAC 地址
- 用户输入

支持 GPS 的手机及专门的 GPS 设备可以使用 GPS。所有能连接到互联网的设备（台式电脑、打印机、路由器等）都可以使用 IP 地址。全世界的手机运营商都可以使用 GSM/CDMA Cell ID。支持无线连网技术的设备则可以使用 Wi-Fi 和蓝牙 MAC 地址。而任何设备都可以通过软件让用户输入自己的位置、邮政编码等信息（一般是通过文本框输入），也就是使用*用户输入*。

1.5.1 GPS

GPS 卫星连续不断地发射信息，支持 GPS 的设备或接收器都可以解析。GPS 发射的信息包括当前工作正常的 GPS 卫星阵列、所有卫星在太空中的大致方位、发射信号的卫星在太空或轨道上的精确位置以及发射信号的时间，等等。接收器通过测定阵列中可见卫星发射信号的时间来计算自己的位置。



要知道在地球上每一时刻有哪些 GPS 卫星可见，可以参考这幅 GIF 动画：http://en.wikipedia.org/wiki/File:ConstellationGPS.gif。

接收器在确定了接收每条消息所花的时间之后，再根据这些信息计算到每颗卫星的距离。每颗卫星到接收器的距离、当前所在的轨道，加上三边测量（trilateration）计算，就可以让接收器知道自己当前所在的位置。虽然在无线电三角测量中，三个发射器足以确定适当的位置，但对于环绕轨道运行的卫星而言，还要考虑时间的因素。卫星信号到达地球是需要时间的，大概几秒钟吧。卫星中任何微小的计时误差再乘上这个时间，都有可能造成极大的定位错误。而第四颗卫星的加入可以消除这种错误（参见图 1-4）。因此，绝大多数情况下，接收器都会使用四个或更多个卫星来计算自己的位置。不过这也不是必需的，在接收器具有已知高度的情况下（比如某些固定的接收器），就可以不使用那么多卫星。

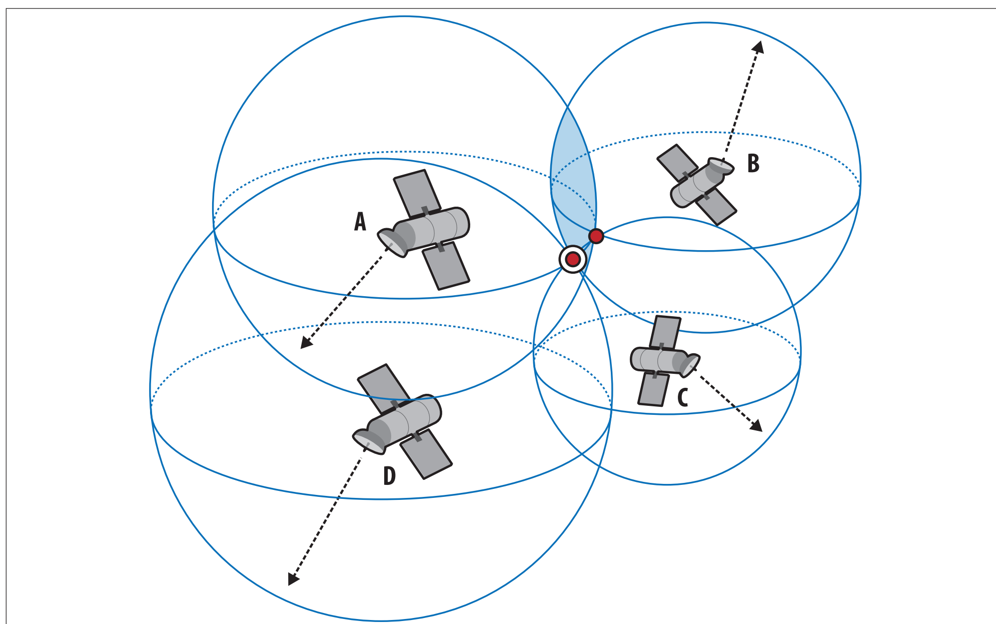


图 1-4　使用卫星的全球定位系统

使用三边测量计算位置

三边测量是通过度量某个点到一组卫星之间的距离来定位的过程。要使用三边测量，首先必须知道要使用的卫星的位置。当卫星向接收器发送信号时，它会发送（1）一个表示何时发送消息的时间戳，（2）一个*星历表*和（3）一个*航空历*。星历表中包含针对特定卫星的轨道信息和时钟校正。航空历也提供包含轨道信息和时钟校正的数据，但针对的是整个卫星阵列。航空历并不十分精确，最长 4 个月才会更新一次，而星历表数据则极为精确，有效期至多 30 分钟。以下是使用 4 个卫星来计算位置的步骤。

1. 取得第一颗卫星（A）的数据，根据距离创建一个球面。接收器位于这个环绕着卫星的球体范围表面的某个位置上。
2. 取得第二颗卫星（B）的数据，测量距离并创建第二个球面。接收器位于两个球面相交处的圆周的某个位置上。
3. 取得第三颗卫星（C）的数据，测量距离并创建第三个球面。接收器位于三个球面相交处仅有的两个交点中的某个点上。
4. 取得第四颗卫星（D）的数据，测量距离并创建第四个球面。接收器的位置由第四个球面与前述两个点中的一个点相交决定。

接收器到卫星的距离是由*光速*和信号离开卫星的时间决定——光速（300 000km/s）乘以这个时间。为了保证测量的精确性，卫星和时钟都精确到纳秒级（十亿分之一秒），而所有现代的 GPS 卫星都内置有原子钟。

1.5.2　IP地址

IP（Internet Protocol，因特网协议）地址是指定给任何上网设备的一个唯一编号，这些设备通过 IP 地址相互通信。这个编号类似于家庭地址。每个家庭都有一个唯一的地址，有了这个地址就可以收到信件，收到外卖，乃至接受紧急情况下的援助。设备连接到因特网时指定给它的 IP 地址，可以用来向其他设备发送数据并从其他设备那里接收响应。家庭地址在某种程度上是固定不变的，而 IP 地址有些是*静态的*（固定不变的），有些则是*动态的*（临时分配的）。无论设备获得的是什么类型的 IP 地址，这些地址总是由被句点分隔的四组数字组成，比如：123.123.123.123。

多数情况下，IP 地址都是通过地区性的注册机构按照地区成批地指定给 ISP（Internet Service Provider，互联网服务提供商）的。正因为如此，通过 IP 地址经常就能知道设备所在的国家、地区，甚至城市。而且，得益于 ISP 近来对数据的收集和维护，根据 IP 地址确定的设备的地理位置经常与其实际位置相差只有数米（参见 1.6 节）。对于想通过 IP 地址来得到地理定位信息的人而言，最困难的就是要面对几百家这种区域性的注册机构，需要一个一个地查询才能得到想要的数据——基本上是不现实的。



近几年来，虽然通过 IP 地址获得的地理定位信息更加精确了，但结果仍然有可能出错，甚至与实际位置差出数英里[4]。查询得到的位置可能是 ISP 自己的位置、代理服务器的位置、防火墙的位置，或者其他向查询设备传送数据的设备的位置。这些位置很可能与实际位置相差甚远。通过 IP 地址查询数据的精确性将在第 3 章讨论。

有很多专门从事收集世界各地 IP 地址范围信息，并将这些信息集合到数据库中供人搜索的公司。这些公司的出现满足了快速查询地理定位信息的需求。这些数据库的建立花了数年时间，但这些公司基于设备的 IP 地址给出的位置信息在今天已经具有了相当高的精确度。我们可随便列举几家，比如 MaxMind、Quova 等公司就提供了基于 IP 的地理定位数据库和 API，供开发人员以免费或按需收费模式访问他们的数据库——IPInfoDB、Geobytes、GeoLite City/Country。而 Geolocation API 则更加方便，开发人员使用它无需查询上述每个数据库，也能根据用户的 IP 地址返回位置信息。当然，上述公司除了能给出用户的位置信息，还提供 Geolocation API 不能提供的其他大量补充信息。

1.5.3 GSM/CDMA Cell ID

Cell ID 是在特定的蜂窝网络中标识每一部移动设备的唯一编号，与在网络中标识设备的 IP 地址很类似。两种最常用的网络是 GSM（Global System for Mobile Communications，全球移动通信系统）和 CDMA（Code Division Multiple Access，码分多址）。你请求的蜂窝服务的类型取决于尝试进入的被相应网络覆盖的区域。

GSM 是最早诞生的一种手机通信技术，因此与市面上的其他同类技术相比具有较高的稳定性。作为一种 2G 通信技术，全球有 200 多个国家和地区支持 GSM 网络，最近的数据表明全球 75% 的手机用户都在使用该网络。从 GSM 迁移到 3G 和 4G 服务（HSPA+，Evolved High Speed Packet Access；LTE，Long Term Evolution；SAE，Service Architecture Evolution）非常简单，因而成了近 40 亿用户使用的标准。尽管 GSM 技术不能像其他技术那样持有相同量级的数据，但由于可以在建筑物内外架设信号转发器以避免接收能力减弱，所以 GSM 服务的质量是相当高的。GSM 因此也就是了一个非常好的载体，很少发生服务中断。

CDMA 是比 GSM 新一些的技术，能同时提供 2G[cdmaOne(TM)] 和 3G[CDMA2000(R) 及 WCDMA] 服务。CDMA 技术的优点在于，它支持多用户在同一频带占用相同的时间和频段，这是 GSM 做不到的。与 GSM 类似，CDMA 网络也正在向 LTE 技术迁移，以便支持用户对 4G 网络的需求。

注 4：1 英里约合 1609 米。

无论手机使用的是什么类型的网络，真正重要的是它们在相应网络中都具有一个唯一的标识符。利用三角测量技术，可以确定 Cell ID 的纬度和经度，从而实现地理定位。参与三角测量移动设备位置的发射塔越多，得到的位置信息越精确。这也是依赖于这种技术的地理定位在城市地区效果更好的原因——与农村地区相比，城区的发射塔更多，相互距离短。



在美国，随着 Enhanced 9-1-1（E911）服务的出现，FCC（Federal Communications Commission，联邦通信委员会）强制要求所有运营商必须达到 95% 的手机能在 300 米的范围内确定自身位置的最低要求。这一强制命令增强了移动设备在蜂窝网络中的地理定位能力。

1.5.4 Wi-Fi和蓝牙的MAC地址

Wi-Fi 和蓝牙的 MAC 地址与设备上的 IP 地址工作方式类似。MAC（Media Access Control，媒体接入控制）地址是指定给接口的唯一编号，通常由接口卡制造商指定。这个编码应该是固定不变且全球唯一的标识符，但比较新的硬件支持手工修改这个地址——这种做法被称为 *MAC 地址欺骗*（spoofing）。MAC 地址的形式类似于：12-34-56-78-9A-BC。

Wi-Fi 路由器的 MAC 地址就是该无线设备接口的地址。同样，蓝牙设置的 MAC 地址也是其接口的地址。使用 MAC 地址的方式与使用 IP 地址差不多，再加上纬度和经度，就可以获知当前设备的位置。

1.6 位置与基于位置的服务

你所在的*位置*或*地点*，就是你的设备在地球表面上由经纬度标出的一个物理空间。但是，地球的表面不是平的。位置通常还有与之关联的其他信息（这一点第 2 章也会介绍到），而这些信息也有助于准确地找出地表上的这个空间的精确位置。

除了获取位置信息之外，还有一些对人们同样有用的信息，比如国家、地区或州、城市、邮政编码、街道地址、时区、域名、ISP、语言、公司名和连接速度等。至于能够得到哪些与经纬度对应的或者说相关的信息，要视取得位置的方法而定（GPS、IP 地址，等等）。

LBS（Location-based Service，基于位置的服务）一般指运行在移动设备上，能够提供客观事实或娱乐性信息的服务。利用地理定位，这种服务可以让客观事实或娱乐的应用对用户而言更加个性化。最典型的 LBS 服务，就是在确定设备所在的位置后，列出该位置附近的所有饭店。随着基于位置的服务越来越深入人心，它们的商业价值

对各类公司也变得更加显而易见。这些公司可以使用 LBS 结合天气、优惠券及广告，为用户提供个性化的服务。这种情况已经较为常见了，今后只能是越发展越快。

基于位置的服务首先要通过可用的方法——GPS、GSM/CDMA Cell ID 或 IP 地址等，来取得设备的位置。在取得由经纬度坐标确定的位置之后，就可以进一步取得事先计划好的其他信息。然后再把这些信息呈现给用户，很可能还会以某种方式与用户交互。

以下是一些流行的基于位置的服务。

- 根据输入的地址提示建议的路线规划
- 提示交通拥堵或事故
- 提示临近商铺、餐馆或其他服务设施的位置
- 与附近的人进行社会化交流
- 出于安全考虑的跟踪家庭成员的应用

这个列表还会继续增长下去，因为今天基于位置的服务可以做的事情是无穷无尽的。基于位置的服务是地理定位的一种最主要的应用形式，但却不是使用地理定位实现其功能的唯一应用形式。建议读者多看一看那些密切关注着 LBS 市场，见证着这个市场日益兴旺发达的站点——LBSZone.com 就是一个不错的起点。另外，Sidney Shek（具有 50 多年历史的技术解决方案提供商 CSC 的撰稿人）写过一篇关于 LBS 的论文，很有价值，短地址为 http://t.cn/amBTNq[5]。

1.7 今天的地理定位

近来（我的意思是从 2009 年前后至今），地理定位这个话题越来越热，手机软件开发人员尤其关注它。基于移动设备特别是智能手机开发的应用与日俱增。这些新软件有一个非常鲜明的特点，它们并不只专注于一个市场，而是涵盖了多种不同的用途。其中有很多应用主要针对社交媒体和交互，但也有越来越多的服务提供了专门基于位置的搜索、实时跟踪和急救服务。

1.7.1 手机应用

手机地理定位应用从 2004 年起就开始兴起了，其标志就是第一个现代社交媒体应用——Yelp 的出现。Yelp 之前也有过其他应用，但 Yelp 的吸引力似乎更广泛——智能手机用户的激增对它的成功起到了一定的作用。从那时起，这个市场已经有了

注 5：长 URL：http://assets1.csc.com/lef/downloads/CSC_Grant_2010_Next_Generation_Location_Based_Services_for_Mobile_Devices.pdf。（译者注）

巨大的发展，包括网络和硬件、提供商和 SDK（Software Development Kit，软件开发工具包），以及导致今天的解决方案处于过剩状态的软件供应商。云解决方案在某种程度上也在加速发展——Esri[6] 与 Amazon Web Service 从 2010 年 2 月开始合作佐证了这一点。

再有，Verizon Wireless、AT&T、Sprint、T-Mobile 等主流网络提供商，都在其设备中内置了某种地理定位应用。除此之外，这些设备的操作系统（iOS、Android、RIM，等等）也都提供了 SDK，供开发人员为这些设备开发本机应用。这一切正是基于位置的服务真正爆发式增长的源泉，特别体现在以下应用的出现：

- 社会化签到应用（Foursqure、Gowalla、Yelp 等）
- 位置共享应用（Shopkick、Glympse 等）

1. 社会化媒体应用

Yelp（www.yelp.com），由 Jeremy Stoppelman 和 Russel Simmons 于 2004 年发布，是第一批主流社交媒体应用，专注于把社区中的人与当地商家联系起来。Yelp 为用户提供了一种查找和评价商家的手段，用户还可以阅读其他人对特定地区商家的评价。同时，Yelp 还允许商家在特价优惠或新品上市时通知用户。Yelp 仍然在继续发展壮大的过程中，“2011 年 1 月统计：过去 30 天有 4500 万人访问过 Yelp。”[7] 基于地理定位将整个社区联系起来是 Yelp 的全部价值所在，很多后来的应用在很多方面都借鉴了 Yelp 的核心理念。

Google Latitude（www.google.com/latitude），是在 Google 的 Maps 应用基础上构建起来的一款地理定位应用。这款应用可以让你在任何时候看到你的朋友所在的位置（当然需要得到他们的许可），你也可以跟朋友共享自己的位置。Google 在 2005 年曾收购了由 Dennis Crowley 开发的一个社交网络应用 Dodgeball，但为了支持 Latitude，2009 年就把这个应用给关闭了。Latitude 分手机和桌面两个版本。虽然在 Android 手机上的功能更全面一些（共享位置、签到、主屏幕微件、隐私保护），但与他人共享位置以及查看朋友所在位置的基本功能，几乎在当前的任何平台中都可以使用。Latitude 是 Google 的一款社交媒体签到应用，但这只不过是各种社交媒体应用的基本形式而已。

Yelp 上线后不久，斯坦福大学两名大二学生 Sam Altman 和 Nick Sivo 就发布了 Loopt（www.loopt.com）。后来，Alok Deshpande、Rick 和 Tom Pernikoff 加入 Loopt，

注 6：美国环境系统研究所公司（Environmental Systems Research Institute, Inc. 简称 Esri）成立于 1969 年，总部设在美国加州 RedLands 市，是世界最大的地理信息系统技术提供商。Eris 中国网站 http://www.esrichina-bj.cn/。（译者注）

注 7：据 About Us | Yelp.（http://www.yelp.com/about）。

专门从事地理定位应用的开发，旨在帮助人们发现自己身边的一切。与 Yelp 相似，Loopt 同样专注于让人们使用这个应用，而不仅仅是商业推广和位置信息，也正在发展成为一个完善的社交媒体应用。

2007 年，Gowalla（www.gowalla.com）登上 LBS 舞台，首创了基于用户的地理定位在“地点”（spots）“签到”（check-in）的概念。由 Josh Williams 和 Sott Raymond 共同创立的 Gowalla，旨在用户通过分享自己的旅游经验、照片和评论，实现社交互动。而且，通过给用户颁发徽章（pins）和奖励以充实“通行证”，Gowalla 也融入了一些社交游戏的元素。最近，Gowalla 也集成了 Foursquare、Facebook Places 和 Twitter 等比较新的 LSB 应用。

SCVNGR（www.scvngr.com），由 Seth Priebatsch 在 2008 年创立，专注于打造一种基于社交网络的游戏平台。玩 SCVNGR 游戏就意味着要去不同的地方，迎接挑战并在此过程中赢得点数。因为 SCVNGR 要做的是一个平台，而非单纯开放式应用、组织、教育机构，任何人都可以创建自己的挑战并将基于位置的奖励直接集成到游戏中。

Foursquare（foursquare.com），曾被 2009 年的 SXSW（South by Southwest Music and Media Conference）奉为“突破性手机应用”，是由 Dennis Crowley（Dodgeball 的作者）和 Naveen Selvadurai 创建的（参见图 1-5）。它从先驱应用中借鉴了经验，一半是游戏，一半是与其他用户分享信息的社交媒体。使用 Foursquare，用户可以在一个地点签到、挣取点数及所谓的“地主”头衔和徽章。用户也可以为自己去过的地方添加提示，以便给其他想到同一地方去的人参考。

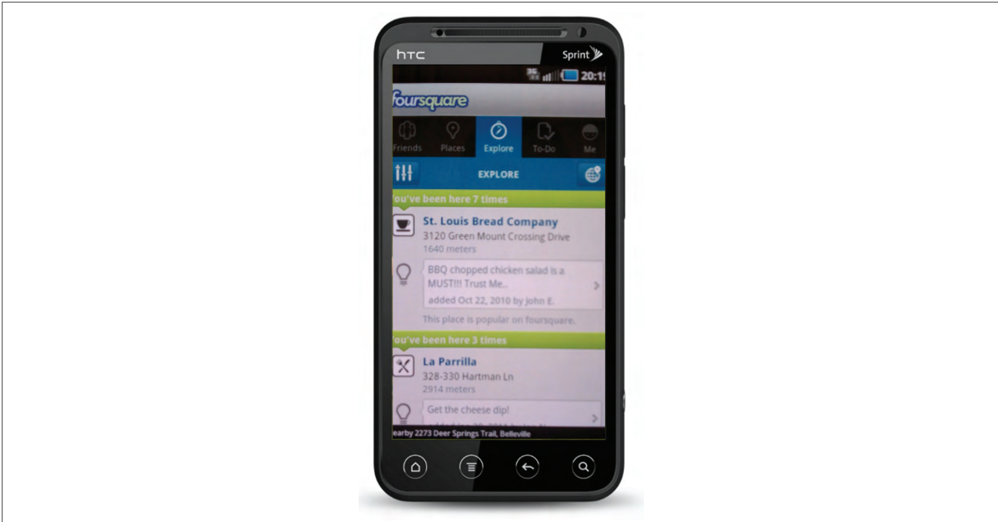


图 1-5　社交媒体应用与地理定位形影不离

2010 年被称为“地理定位年”，确实如此。Foursquare 的用户群急速扩大，在本书写作时已经达到 750 万人，成为当前所有地理定位应用中最炙手可热的一个。然而，人们不禁要问：下一个会是谁？社交媒体巨人 Facebook 在 2010 年发布了自己的 LBS 服务——Facebook Places，考虑到它 5 亿多的活跃用户，地理定位的大爆发指日可待。地理定位应用的社交媒体角色在今后几年中还将进一步加强。

2. 位置共享应用

Glympse（www. glympse.com），是 2008 年由三位前微软员工 Bryan Trussel、Jeremy Mercer 和 Steve Miller 共同创立的，通过它能以不一样的方式实现与他人共享位置信息。与 Google Latitude 类似，Glympse 规定只有那些被选中的人才能看到别人的位置信息。它与所有其他地理定位应用的差别就在于其信息共享方式：不依赖用户在各个地点签到和其他人所共享的更新，而是使用运行应用的设备内置的 GPS 来实时地在网页地图中显示用户的位置。共享的时长也由预先设置好的时间长短决定，除了启动“glympse”之外，不再要求用户有什么交互。

Shopkick（www. shopkick.com）是由 Cyriac Roeding、Jeff Sellinger 和 Aaron Emigh 在 2009 年创建的，目的是利用手机为人们创造一种全新的购物体验。在与美国部分最大的零售商合作的基础上，它让用户只需走进商店就能获得奖励和特别优惠。它不需要用户在走进商店时签到，因为 Shopkick 会使用 GPS 来检查用户与签约商店的最近距离（在给定的误差半径内），在他们走进商店时自动签到。

位置共享应用的市场还将继续增长，因为无需跟应用交互就可以与他人共享自己的位置是一种更好的方式。这种应用的最大问题就是隐私保护。对于这些 LBS 应用，如果能够让用户觉得它们没有多少社交功能，而是像 Shopkick 这样属于面向服务的产品，就会有更多有新意的地理定位方案出现。

1.7.2 增强现实应用

增强现实是真实的视图（通常来自摄像头或其他镜头）与计算机基于传感器输入生成的视图的组合。比如，把手机的摄像头对准一条城市街道，增强现实应用就会显示它通过镜头“看到的”与这条街道、汽车、人流、建筑物和天气等相关的地图或文本信息。所有这些信息都被放在实际拍摄到的画面上，为用户提供大量实时的辅助信息。

目前移动设备上的增强现实应用使用地理定位作为一种辅助手段，以便确定什么时候应该为用户提供哪些信息。此外，为了让增强现实的内容更丰富，移动设备还会利用其内置的各种传感器，以便生成更多的信息。这种应用的市场方兴未艾，虽然目前还没有太多案例，但这种技术在未来的应用一定会越来越多。

Layar（www.layar.com），2009 年由荷兰阿姆斯特丹的 Claire Boonstra、Maarten Lens-FitzGerald 和 Raimo van der Klein 创建，是一个手机增强现实平台，能够根据镜头拍摄到的画面显示各种不同的信息。比如天气预报、房地产、政府部门、餐馆、旅游和娱乐场所等。这些可以看到的信息叫做图层（layer），在一般的浏览器中以网页形式呈现。2011 年，图层技术被世界经济论坛和《时代》杂志称为技术先驱。

而 acrossair Augmented Reality Browser（http://t.cn/au6La9[8]）则是一款浏览器应用，可以搜索 Google 或维基百科，从 Flickr 及 Panaromia 等站点获取图片，访问 Twitter 或 Yelp 等社交媒体——全部内置于一个为摄像机增加现实信息的应用中。自从 2009 年在苹果商店上架以来，acrossair 就一直没有停止过对其进行改进，致力于把它打造成导航辅助应用。这是个一站式的应用，通过它能够找到与你的兴趣点相关的几乎一切信息。

Yelp Monocle 也是一个增强现实的服务，于 2009 年加入到目前的 Yelp 社交媒体中。它利用手机的 GPS 和陀螺仪，根据手机的朝向，在拍摄到的视图上以标签形式显示附近的商家信息，如图 1-6 所示。这些标签从 Yelp 的主业务数据库中获取信息并显示客户的评级、到每家的距离、商家类型，而且在有数据的情况下，还会显示商户的营业时间。



图 1-6　Android 平台上的 Yelp Monocle

从目前能够看到的这些增强现实应用的早期案例，我们有理由相信还会有更多的解决方案出现在这个特定的市场中。作为一种比较前沿的利用地理定位的技术，增强现实一定会在不久的将来支撑起一批真正令人惊叹的应用。

注 8：长 URL：http://itunes.apple.com/cn/app/acrossair-augmented-reality/id348209004?mt=8。（译者注）

第 2 章

地理定位：不仅是经纬度



第 1 章介绍了地理定位的基本情况，包括地理定位的含义、不同的设备如何从不同的来源取得地理定位信息，以及当前常见的一些地理定位应用。在此期间，也提到了一些基本术语，比如纬度、经度和海拔高度，但却没有对这些术语给出定义。有的读者可能对 GIS 的术语已经非常熟悉了，但万一你还不是很了解，那本章将为你解读一下 W3C Geolocation API 需要开发人员熟悉的相关信息。内容应该熟悉，只有知道了如何适当地使用它们，才能创造出更好的应用来，才不会把错误的数据传递给最终用户。

2.1 坐标系统

前面提到过，设备的位置（地点）是使用 GPS 或其他定位方法发现并以经纬度坐标形式给出的。换句话说，我们取得的是特定设备的*坐标*（coordinates）。为了定位地球上的设备，可以用一组数字来表示它在地球上的位置。这些数字构成了我们可以据以推测的系统。

在数学和日常生活中，有很多种坐标系统。实事上，当我们最初学习正负数的时候就已经接触了最基本的坐标系统——*数轴*。学习过其他数学课程的读者可能还了解其他坐标系统，比如*笛卡儿坐标系*（x、y 和 z）和极坐标系（r、θ）。具体到地理定位，使用的则是*地理坐标系*。在地理坐标系中，坐标由纬度、经度和海拔高度来表示。

经度和纬度

要理解经度和纬度，可以画一个球体，然后在球面上按（大致）相同的间隔画出垂直和水平方向的线，如图 2-1 所示。之所以说是大致画出等分线，是因为 2.2.1 节将会介绍的，地球并不是完美的球面，所以水平线之间的间距会有一些小的变化。这种经纬度系统最早可能是由埃及人发明的，但画出这些线的人则是公元前 3 世纪希腊的天文学家、数学家和地理学家埃拉托色尼（Eratosthenes）。后来，亚历山大的学者又利用*角度*（度，“步”）将地球分割成有序的网格。[1]

球体上的水平线是*纬线*，两条纬线的间距大约 69 英里（111.04 公里）。赤道的纬度是 0 度，南、北两个半球的纬度从赤道开始向两极递增，最大为 90 度。北极点对应的是北纬 90 度，南极点对应的是南纬 90 度。纬度用 φ（小写的希腊字母，音“佛爱”）表示。

注 1：参见 Tom Garrison 著的 *Oceanography: an invitation to marine science*，Cengage Learning，2007 年。

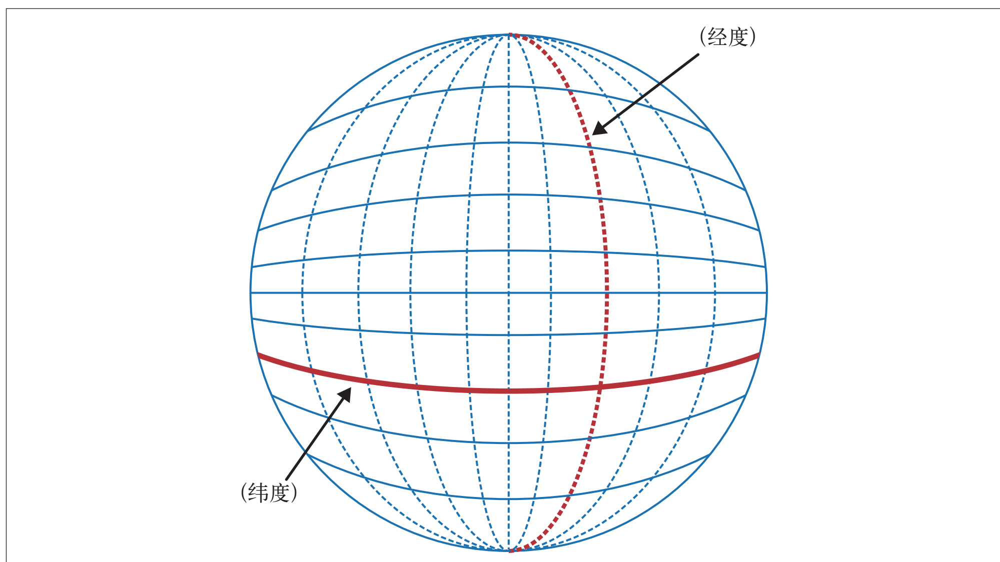


图 2-1　地球上的经、纬度示意图

球体上的垂直线是经线，这些线在南极点和北极点汇合，在赤道上达到最宽的间距（也是大约 69 英里或 111.04 公里）。经线也称为子午线，本初子午线（也就是零度经线）是 1884 年确定的经过英国格林尼治的经线。以本初子午线为起点，东西半球的经线各划分为 180 度。经度使用 λ（小写的希腊字母，音“兰乌达”）表示。



1884 年之前，任何航海的国家在发行航海图的时候都可以设定自己的“零度经线”。这种现象持续了近几个世纪。实际上，这个传统仍然起源于亚历山大学者将埃及的亚历山大作为“零度经线”。

1. 十进制度数与度分秒

在定位地球上某一点时，为了达到必要的精度，经纬度实际会被分成度（°）、分（'）和秒（"）。1 分等于 60 秒，1 度等于 60 分。举个例子，密苏里州圣路易斯拱门（St. Louis Arch）[2] 的坐标是 38°37'29"N, 90°11'7"W，即北纬 38 度 37 分 29 秒，西经 90 度 11 分 7 秒。

在以度数表示的坐标中，经常能够看到如下形式：

- 度、分、秒（及秒的小数部分）

注 2：圣路易斯拱门座落在密西西比河畔，高 192 米，是著名建筑设计师伊洛 · 萨里嫩于 1940 年在一次设计竞赛中设计的。直到 1963 年 2 月才开始动工，1965 年 10 月完工。被人们誉为通往美国西部的大门。

- 度、分（及分的小数部分）
- 十进制度数

拱门的坐标是以第一种形式即 DMS（degrees，minutes,seconds，度分秒）形式表示的。第二种形式（度和秒）并不常见，但也是有效形式。第三种形式（十进制度数）是将 DMS 坐标的分和秒都转换成度的小数形式的表示方式。十进制度数与另外两种形式的不同点在于，它不使用主方向（北、南、东、西）来表示经纬度的方向，而是使用正负号，比如：

38˚37'29"N	38.624 722
56˚12'13"S	–56.203 611
124˚11'7"W	–124.185 278
12˚57'24"E	12.956 667

2. DMS到十进制度数的转换

把 DMS 坐标转换成十进制度数很直观，只要按照如下步骤做即可。

(1) 计算总秒数
(2) 用这个总秒数除以 3600（每度的秒数）
(3) 将得到的小数添加整数度数后面
(4) 如果坐标是南纬或西经，则为结果加上负号

下面就按照这几步来转换一下拱门的经度坐标（90˚11'7"W），看一看计算过程。

(1) 计算总秒数：11'7" = (11 × 60 + 7) = 667
(2) 用这个总秒数除以 3600：(667 / 3600) = 0.185 278
(3) 将得到的小数添加在整数度数后面：90 + 0.185 278 = 90.185 278
(4) 因为是西经，所以为结果加上负号：–90.185 278（表示格林尼治以西）

3. 十进制度数到DMS的转换

很简单，对吧？从十进制度数坐标转换到 DMS 坐标也很直观，只要按照下列步骤做即可。

(1) 从整个坐标值中减去整数，得到小数
(2) 用小数乘以 60（得到分）
(3) 从整个分的值中减去整数，得到小数
(4) 用小数乘以 60（得到秒）
(5) 如果十进制坐标是负的经度值，那么保留整数度数的符号，或去掉符号并加上一

个 W；否则，加上一个 E。如果十进制坐标是负的纬度值，那么保留整数度数的符号，或去掉符号并加上一个 S；否则，加上一个 N。

可能这里的方向会让人觉得有点困惑，不过看一个实际的例子就会比较清楚。以 –90.185 278（经度）为例。

(1) 从整个坐标值中减去整数，得到小数：90.185 278 – 90 = 0.185 278
(2) 用小数乘以 60（得到分）：(0.185 278 × 60) = 11.116 68
(3) 从整个分的值中减去整数，得到小数：11.116 68 – 11 = 0.116 68
(4) 用小数乘以 60（得到秒）：(0.116 68 × 60) = 7.000 8（舍去 8，得到 7）
(5) 原来的坐标值是负经度值，因此去掉符号并加上 W：90˚11'7"W

有读者可能不理解，为什么要在这里讲这些换算过程呢？等到第 3 章我们讨论 W3C Geolocation API 对位置请求给出的返回值时，我们的用意就会不言自明了。请大家相信，这个换算过程很重要！

2.2 大地测量系统与基准点

如果一切就那么简单，使用地理坐标系统来表示某一点在地球上的位置已然足矣。但由于地球的形状不规则，问题就没有那么简单了。为此，需要一个参照系统来将地图上的点转换成地球上的实际位置。这个系统就叫做*大地测量系统*（geodetic system）。大地测量系统用来转换坐标的参照物叫做*基准*（datums），而*大地基准*（geodetic datums）就是测绘和大地测量中使用的参照物。



大地测量学是地质学的一个分支学科，研究地球的形状和地图上某一点实际位置的确定。大地测量、测地指的就是这种度量[3]。

大地基准用来确定地理坐标系统的位置，固定它的原点，定义地球的形状。从地理定位的角度说，大地基准的意义在于它能把地球描述为一个平面——要显示三维的地球并不容易，特别是在 Web 上。就拿 Google 举个例子吧，打开 http://maps.google.com/，就会显示一个已经转换为平面的地球。而打开 http://earth.google.com/，就会显示一个三维的地球。这两种地图使用不同的基准来显示相同的数据。

2.2.1 地球的形状

前面已经多次提到地球的形状，它如何影响地理坐标系统和纬度，也提到了需要大地基准测量的原因。那么它到底哪里不圆呢？

注 3：普林斯顿大学“About WordNet”WordNet，普林斯顿大学，2010 年。http://wordnet.princeton.edu/。

我们上学的时候都学过，公元前 6 世纪的古希腊人就已经知道地球是圆的了——当时叫做毕达哥拉斯球体。数个世纪以来，数学家和哲学家不断改进自己的理论，不停地进行实验，他们得到的结果也越来越接近我们今天所知道的地球的形状。但有一个问题，我们可能没在课堂听老师说起过——如果在自己年长一点儿的时候听过更高级的自然科学课，可能你已经知道了：地球并不是一个真正的球体。

没错，球体很接近地球的形状，而且也将作为很多应用（包括 Google Earth）中映射的模型。但是，随着人类对重力场的深入研究，再加上更丰富的地理数据（特别是在人造卫星的辅助下取得的数据）作为支撑，我们发现地球在赤道处略鼓，而两极稍扁。因此更确切地说，地球应该是椭球体，如图 2-2 所示。正因为地球实际上是椭球体，所以就需要有一个基准来把地球简化成两维或三维的模型，以便人们日常使用。

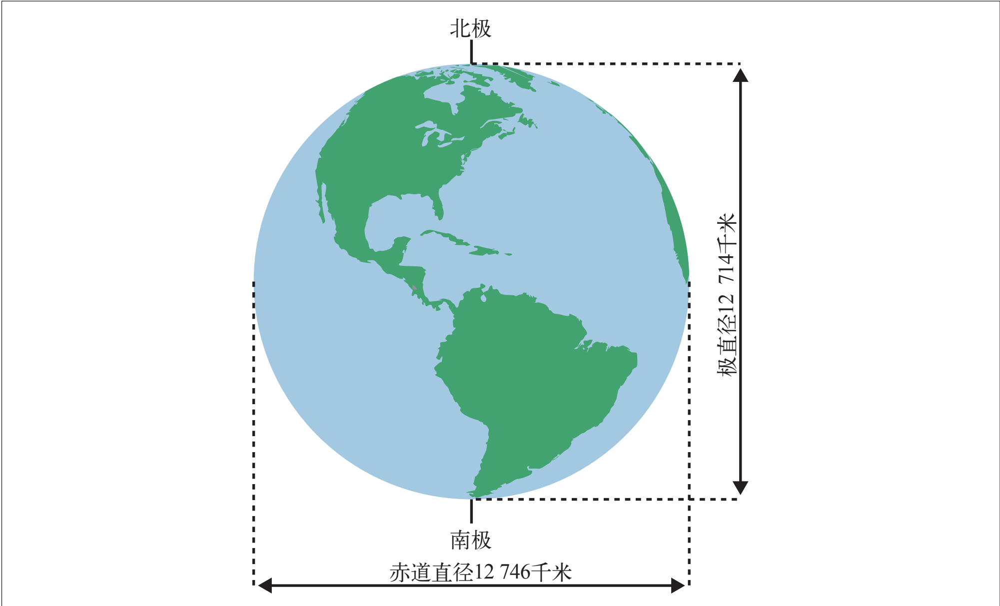


图 2-2　地球是椭球体

2.2.2　公共基准

数学家在不断修改和重新计算地球的形状和大小的同时，也在不断地修改所使用的基准。不同时期有不同的基准，这些基准随着时间推移，已经从把地球描述成一个球体，逐渐变为将其描述成与实际情况最为接近的椭球体。

既然地球不是完美的球体，那么不同地区使用不同的基准，就比统一使用覆盖全球的基准来得更精确。以下就是几个今天在用的比较常见的“全球”基准：

- 世界大地坐标系（WGS 84）
- 北美大地基准（NAD 83）
- 欧洲基准（ED 50）

与之相对，以下是全球在用的许许多多更加本地化的基准中的几个例子：

- 英国国家测绘局（OSGB 36）
- 瑞士基准（CH 1903）
- 日本基准（TOKYO）
- 普尔基准（S-42）

任意两个基准之间的坐标差叫做*大地基准偏移*（datum shift），是查看地图时的一个重要依据。基准之间的偏移量取决于很多因素，包括相关基准的海拔高度。比如说，虽然 WGS 84 与 NAD 83 之间的差别非常小（平均不超过 70 米），但 WGS 与 OSGB 36 之间的差异则大约是前者的两倍（140 米左右）。从地球这个庞然大物的角度来看，这些差异确实微乎其微。但如果要在一小块地方确定坐标，那 140 米恐怕就要差出几个城市街区了。

WGS 84

世界大地坐标系（World Geodetic System）是由美国国防部和世界各地的科学家创立于 1960 年（WGS 60）的一个全球基准。诸多因素都决定必须有一个全球性的基准，以便供其他地区性基准参照。其中最重要的一点，就是当时已经存在一些大地基准（NAD 27、ED 50 等）还未达到作为全球性基准的精确度要求。而且，随着国际贸易和全球旅游业的发展，加上科学项目的蓬勃发展，对世界地图的需求日益迫切。自从创立以来，世界大地坐标系几经修订，先后出现了 WGS 66 和 WGS 72。1984 年又修订和发布了 WGS 84，2004 年对这个版本又进行了一次修订。

WGS 为大地测量提供了三个关键组件：一个用于在地球上确定坐标的框架、一个定义了地球表面（全球平均海平面）在数学上理想状态下的*大地水准面*，还有一个参照球体。当然，WGS 的参照球体本身是椭球体，在这个椭球体上可以叠加经纬度坐标系统。2004 年修订 WGS 84 时，在计算地球重力模型（EGM 96）的基础上，推出了一个新的大地水准面。在此之前，使用的都是 EGM 84。



随着地球自转，其重心不断改变，间或地更新全球基准的大地水准面是必需的。比如，根据美国宇航局喷气式推进实验室 Richard Gross 的计算，2011 年日本本州的 9.0 级地震致使海平面上升了 0.22 米，洋底的偏移足足导致一天缩短了 1.8 微秒。而 2010 年智利的 8.8 级地震也将海平面抬高了 0.16 米。

2.2.3　地图投影

要生成世界地图，无论是纸质地图，还是显示在浏览器中的地图，都必须把地球由三维形式投射为两维的形式。对于地球上比较小的地方，这样投影没有问题，但要产生整个地球的平面图，要解决的问题可就多了。以大地基准作为参照，可以通过投影显示地球的某些方面：比例、距离、区域、形状，等等。然而，任何地图都不可能同时精确地显示所有方面；换句话说，要想精确显示某些方面，必须牺牲另一些方面的精确度。

在 Web 领域，所有主要的地图数据提供商（Esri、Google、Microsoft 等）使用的都是基于*墨卡托投影法*（Mercator Projection）的地图投影。墨卡托投影法因以比利时地图制作师杰拉杜斯 · 墨卡托（Gerardus Mercator）而得名，墨卡托在 1569 年绘制了圆柱形的地球投影图。按照这种投影法，所有经线与纬线呈 90 度角交叉，地球上的地理方位在与赤道接近的地方保持正常，而在两极地方则有很大的偏斜，如图 2-3 所示——格陵兰岛的实际大小不会接近非洲大陆的大小。

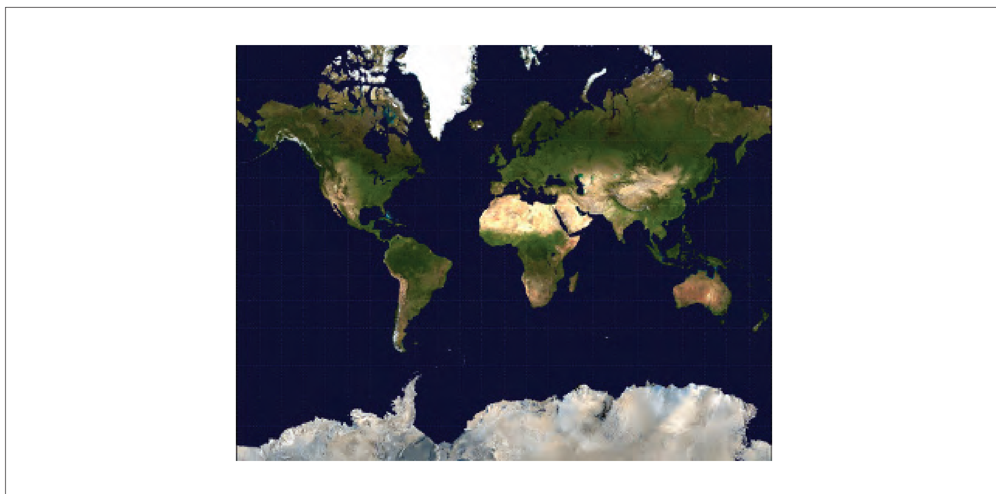


图 2-3　使用墨卡托投影法绘制的世界地图

Web Mercator Auxiliary Sphere 是互联网业内广泛采用的生成 Web 地图的一种投影法。简言之，Web 上使用的地图是以 WGS 84 为大地基准，使用墨卡托投影法生成的。

2.3　高度、路线与速度

如果非要对地理定位的属性按照重要性进行排名，那么位置的经纬度显然是最重要的。但我前面也提到过，在地理坐标系统中，还有第三个属性也同样重要：*高度*

（也称海拔或标高）。除了这三个最重要的属性之外，还有一些附加的属性，尽管有时候不是必需的，但有时候对地理定位应用也会非常有用。这些属性彼此之间具有相关性，适用于移动中的被定位物体，它们是：路线和速度。

2.3.1 大地高度

如果没有海拔数据，那么就不能基于地形特征来确定某个点的位置。所谓海拔，在谈及地理定位时，一般的理解就是指海平面之上的高度，但实际上这不是海拔的确切定义。随着技术的进步，各种各样的环境纷至沓来，使用地理定位不仅能够发现海平面以上的东西，照样有可能发现海平面以下的东西。事实上，在大地基准中，海拔可以有两种表述方式：海平面和测地学。

垂直大地基准

在度量陆地上某一点的高度时，通常的基准是地球的平均海平面（MSL，Mean Sea Level），如图 2-4 所示。通过长期观测海平面的高度，就可以计算出平均海平面的值，其中剔除了潮汐等海洋现象。但是，地球上不同地区之间的重力差还是会对关于垂直大地基准的平均海平面产生影响。正因为如此，有些国家会选择某个特殊的点作为其标准平均海平面，用作绘制地图时的参考点。比如在加拿大、美国和墨西哥，这个地区性的点就在加拿大的魁北克省。

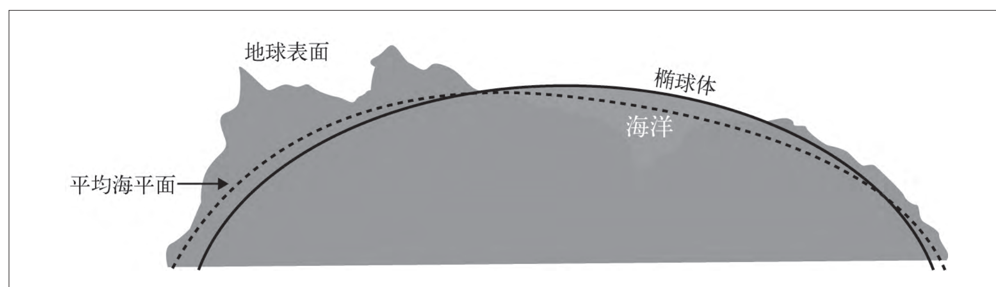


图 2-4　地球上任何一点的高度都不相同

有些情况下，平均海平面也不是垂直大地基准的最理想参照，比如在标绘某些涉及历史遗留问题的地形时，这个问题就会出现。况且海平面也不是固定不变的，因此在这些情况下就要使用一个不同的垂直大地基准。大地基准不会考虑海平面，它只是将地球表面上的任意一点的海拔高度指定为“零高度”。这一点通常与国家在定义平均海平面时指定的地区点是一致的，但不同国家之间的“零高度”点可能就会不一致。比如，北美地区使用的 NAVD 88，就是一个大地垂直基准，碰巧该基准与作为平均海平面的魁北克省的地区性点是同一点。

2.3.2 路线

物体的*路线*指的是它从一个点移动到另一个点的预期安排。这个路线可以使用两点之间事先确定的通路来定义，也可以使用两点之间的必经之路来定义。路线是由两个点之间的直线组成的，路线中的每一个这样的片段叫做*腿*（leg）或航段。

物体的*方向*（heading）指的是物体在任意时刻所对准的方向。方向以角度表示，这个角度是相对于一个固定的参考点（多数情况下是真北）来计算的。计算方法为顺时针从 0° ~ 360°，其中：0° 表示北，90° 表示东，180° 表示南，270° 表示西。

路线和方向有时候也可以互换使用，但细究起来它们还有差别的。如前所述，方向指物体*朝向*哪个方位，但不一定会朝该方位移动。而路线呢，表示的是*预期*的移动方向（可以想一想在确定旅行线路时"在地图上标出路线"来）。于是，在谈到方向和路线时，就又有了另一个术语：*足迹*（track）。足迹就是指从起点（开始移动时所在的点）到当前点的方向。或者还可以换一种说法——足迹是已经走过的路线。

2.3.3 速度

本章到目前为止所讨论这些概念，没有一个比速度更好理解的。看到速度这两个字，你一定会联想到"物体移动得有多快？"，而这也正是在地理定位的背景下这个概念的含义。速度表示的就是物体移动的快慢。这个概念很好理解，但我不希望后面的一点物理解释把你搞晕——我相信不会。

从物理或数学的角度看，对速度还可以给出更准确的定义，即速度是物体*速率*（velocity）的*量级*（magnitude）。物体的速率指的是物体的位置在给定方向上变化的速度。速度和速率都要考虑物体移动的距离和移动该距离所花费的时间，度量单位是"米每秒"（国际单位制规定的标准单位）。

计算物体平均速度的方法如下：

$$V = d / t$$

其中，d 是移动的总距离，t 是花费的总时间。

但不要把它跟物体的*瞬时速度*，或者任意时刻的速度（v）混淆起来。瞬时速度等于距离 s 的导数，其中 s 是单位时间 t 所移动的距离。

$$v = \mathrm{d}s / \mathrm{d}t$$

尽管国际单位制规定的速度单位是米每秒，但我们日常生活中（特别是用于描述汽车的速度时）用得更多的还是米每小时和公里每小时。

2.4 精确度

理想情况下，发送给地理定位请求的位置信息应该准确无误——准确地标识你所在的位置。当然，理想情况是不存在的（如果我说出实情让你失望了，那很抱歉）。在现实中，每一次地理定位请求返回的数据的精确度都是不一样的。只要冷静地想一想地理定位背后的工作机制就会明白，取得位置信息要涉及那么多因素，而我们得到的信息居然还那么精确，这的确有点令人不可思议。

在继续讨论之前，我们先来给*精确度*下个定义。地理定位的精确度，指的是位置信息与物体实际位置接近的程度。在 GIS 中，经常能听到“某点的精确度在 20 米之内”这样的表达方式，意思就是物体实际的位置距离我们给出的位置不会超过 20 米远。图 2-5 展示的是一个带有不同精确度半径的点。

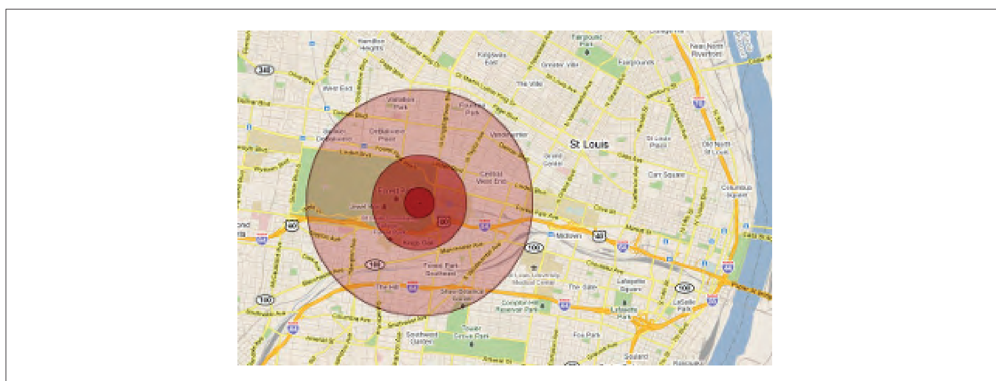


图 2-5　地图上某一点的精确度半径

影响地理定位精确度的一个主要因素是取得位置的方式。IP 地址不如 Cell ID 精确，而 Cell ID 不如 GPS 精确。为什么呢？之所以 IP 地址最不精确，就是因为其位置信息是通过路由器或防火墙的 IP 地址来确定的，而路由器和防火墙很可能距离发送地理定位请求的计算机浏览器有几英里远。这种情况在大公司里是非常常见的。Cell ID 一般会比 IP 地址精确一些，因为三角测量必须根据基站来计算才能得到地理定位。GPS 通常又比 Cell ID 更精确，因为计算的依据是太空中的众多卫星，而且计算过程也更复杂。当然了，硬件故障、无线电干扰、天气状况等因素随时随地都可能导致信号损失，造成精确度下降。而这也是在发送地理定位请求时，取得精确度信息的重要意义，这样才能告知用户他们在应用中的位置存在多大的误差。随着技术的进步，地理定位信息的精确度将会越来越高。然而，即便是最可靠的技术解决方案，也总会受制于一些供应商无法控制的因素。

第 3 章

地理定位API编程



此时此刻，我想一定有不少读者在心里暗自抱怨："作者先生，非常感谢你为我介绍了那么多有关地理定位的背景知识，但什么时候能让我们看一看怎么写代码来取得位置信息呢，现在可以了吗？"假如我说到了你的心坎上，那你太走运了，这一章就要讨论如何使用 W3C Geolocation API 来编程了。

前几章介绍的背景知识与我们要讨论的 Geolocation API 有莫大的关系。比如说，知道我们在用户浏览器中取得的经度和纬度是基于 WGS 84 基准的会非常有用。如果你不明白这句话的意思，还是回过头来看看（或重新看一看）第 2 章吧，这样你才能真正理解本章所要讨论的内容。

3.1 W3C Geolocation API

W3C Geolocation API 是一个规范，规定了如何使用脚本访问主机设备的地理位置信息。[1] 这个规范的目的是提供一套"高层次接口"，让开发人员无需知晓如何确定位置信息即可使用它们。有了这套 API，设备在地理定位时是使用 GPS、IP 地址，还是 Cell ID 都不重要了，重要的只有地理定位信息本身。但是，这个规范也有一个问题，即它不保证 API 返回的位置信息是设备的真实位置。这一点其实无需大惊小怪，有可能 GPS 缺少可见的卫星来确定准确的位置，有可能基于 Cell ID 进行三角测量时没有足够的基站，IP 地址也有可能因为被人修改过而返回一个完全错误的位置。由于存在这样或那样的不确定性，开发人员可以适度地信任由该 API 返回的结果，但决不应该完全依赖它。



最新版的 W3C Geolocation API 规范于 2010 年 9 月 7 日成为 W3C 的候选推荐标准。

3.1.1 当前的API支持情况

目前，大多数最新版本的桌面和移动浏览器都支持 W3C Geolocation API。表 3-1 给出了当前的 API 支持情况。开发人员需要考虑的最大问题是版本老一些的浏览器并不支持这套 API，因为这套 API 是在那些浏览器发布之后才制定的。而对开发人员来说最大困难则是 IE 8（甚至还有 IE 7）仍被广泛使用，包括还有很多手机用户并没有把软件（或硬件）升级到最新版本。不过，好消息是今后发布的所有浏览器和手机都应该支持这套 API。

注 1：Geolocation API Specification: W3C Candidate Recommendation 07 September 2010。编辑：Andrei Popescu，Google, 股份有限公司。 http://www.w3.org/TR/geolocation-API/。

表3-1　支持W3C Geolocation API的浏览器及版本

Web浏览器	支持的版本
Firefox	3.5+
Chrome	5.0+
Internet Explorer	9.0+
Safari	5.0+ [a]
Opera	10.6+
iPhone	3.1+
Android	2.0+
黑莓（BlackBerry）	6+ [a]

a 已经正确实现，但没有完全实现。

3.1.2　其他浏览器的解决方案

如前所述，并非所有浏览器都支持 W3C Geolocation API，不支持的“老古董”们永远不可能提供本机支持。好就好在，已经有程序员拿出了自己的解决方案，他们编写了一些包装库，能够为这些浏览器增加大部分 Geolocation API 的功能。可是，这些库之间也存在一些差异，要想使用它们写出在所有浏览器中都能够正常运行的代码，恐怕还是要多费一番心思。首先，我们来介绍其中几个解决方案，然后再讨论一下怎么处理它们之间的差异。

1. Gears

Gears 以前叫做 Google Gears，是由 Google 公司的一些牛人编写的一个代码库（看它的名字就知道出处）。Gears 从一开始就是一个开源项目，致力于为浏览器增加新的功能，以便开发人员能够开发出更强大的 Web 应用。Gears 的所有组件中，我们最感兴趣的是它的 Geolocation 模块，这个模块提供的功能与 W3C Geolocation API 很相似。实际上，W3C Geolocation API 中有一部分很可能就是模仿的 Gears。



2011 年 3 月 11 日，Gears API Blog 宣布，Google 不会再发布 Gears 的新版本，也不会在比较新的浏览器（如 Firefox 4 和 IE 9）中对 Gears 提供支持。而且，将在 Chrome 12 中去掉 Gears。不过，老版本的浏览器仍然可以继续使用 Gears 获得缺少的功能。

使用下面这行代码就可以把 Gears 添加到网页中：

```
<script type="text/javascript"
   src="http://code.google.com/apis/gears/gears_init.js"></script>
```

至于在代码中使用 Gears 时有什么需要特别注意的问题，现在还不需要担心。后

面马上就会向大家介绍另一个库，利用它可以隐藏 Gears、其他 API 和 W3C Geolocation API 之间的差异。第 4 章还会向大家展示更多这方面的示例，而现在只要知道有这么一个库可以随时使用就够了。

2. 其他手机中的API

当然，为了让开发人员在他们的设备上使用地理定位，硬件供应商都有自己的规范和 API——特别是在地理定位还没有流行的时候出现的一些早期设备上。以下列出了一些提供这种专有 API 的平台，根据连接到你的站点的设备不同，有时候必须依赖于这些平台：

- iOS
- 黑莓
- 诺基亚
- webOS（Palm）

在面向所有 OS 版本低于 3.0 的 iPhone 上开发基于位置的应用时，都要依赖苹果公司自有的 Geolocation API。黑莓开发人员也一样，在面向 OS 版本低于 6 的黑莓设备开发时，也要依赖黑莓自有的实现。

诺基亚也有自己的 Geolocation API，能够在其随机浏览器中运行，就如同 Palm 在其 webOS 2.1 SDK 中添加的 HTML5 Enhancements 模块一样。希望这些供应商将来都能够提供完全兼容 W3C Geolocation API 的浏览器或支持，从而消除开发人员当前所面临的困难。

这样，我们就有了……

3. geo-location-javascript

geo-location-javascript 是一个试验性的框架，尝试将所有底层平台的实现包装成一套与 W3C Geolocation API 规范类似的 API。这套 API 目前支持以下平台：

- iOS
- Android
- 黑莓 OS
- Gears
- 诺基亚 Web Run-Time（WRT）
- webOS Application Platform（Palm）
- Torch Mobile Iris Browser
- Mozilla Geode

这套 API 有两个主要功能：检查连接的设备是否具备地理定位能力和取得设备的位置。正如 3.2 节即将介绍的，这两个功能只是 W3C Geolocation API 全部功能很小的子集，如果你想添加更多功能，必须自己编写代码。

看一看下面这段代码：

```
<script type="text/javascript"
    src="http://code.google.com/apis/gears/gears_init.js"></script>
<script type="text/javascript" src="geo.js"></script>
<script type="text/javascript">
  function initialize() {
    if (geo_position_js.init())
      geo_position_js.getCurrentPosition(show_position, error_handler,
         { enableHighAccuracy: true }
      );
    else
      alert('Geolocation functionality is not available.');
</script>
```

代码的第一行把 Gears 加载到应用中，第二行加载的是 geo-location-javascript API。在真正的应用中，应该在文档的 `onload` 事件发生时调用 `initialize()` 函数。

`initialize()` 函数很好理解，它先检查设备是否支持地理定位，如果支持就取得设备的位置。如果设备不支持地理定位，则显示消息告知用户该结果。与其他 Geolocation API（包括 W3C Geolocation API）一样，geo-location-javascript API 也要求用户事前同意（opt-in），即在继续之前准许应用获取位置信息。3.8 节有关于用户隐私的详细介绍。

调用的 `geo_position_js.getCurrentPosition()` 函数有两个回调：`show_position` 和 `error_handler`。如果在取得设备位置期间遇到了问题，或者用户事前不同意搜索位置，就会调用 `error_handler` 函数；否则，就会调用 `show_position` 函数，之后应用就可以使用设备的经度和纬度数据了。



如前所述，geo-locatioin-javascript API 不像 W3C Geolocation API（下一节将详细讨论）那样支持轮询位置方法。因而你需要自己使用 JavaScript 的 `setInterval()` 方法重复调用 `getCurrentPosition()`。

3.2 功能更完备的W3C Geolocation API

本章乃至本书的重点，是介绍如何在 HTML5 应用中使用 W3C Geolocation API 的 JavaScript 实现。所有最新的桌面和移动浏览器都包含了这个实现。但这个 API 毕竟还比较新，如果要得到一个同时支持其他版本较老的设备和浏览器的跨浏览器实

现，还必须借助其他 API。上一节介绍的 geo-location-javascript 就是这样一个 API。

使用这些 API 有一个缺点，即它们不具备 W3C Geolocation API 的全部功能，很多地方都需要开发人员另外编写代码。看完本章剩下的内容你就会发现，W3C Geolocation API 是一个非常完备和十分强大的接口，利用它可以在设备本机上开发出堪与竞争应用匹敌的 Web 应用。

3.3 Geolocation对象

W3C Geolocation API 规定了与 `Geolocation` 接口相关的对象、属性及方法的通用实现。其中，有一个对象持有 W3C Geolocation API 的全部实现，它就是 `Geolocation` 对象。在 JavaScript 中，可以使用 `Geolocation` 对象来获取浏览器所在设备的地理定位信息。`Geolocation` 对象是浏览器对象 `window.navigator` 的一个新属性，可以这样来访问它的实例：`window.navigator.geolocation`。

与使用其他 JavaScript 对象一样，使用 `Geolocation` 对象之前，最好先测试一下浏览器是否实现了这个对象，如下所示：

```
if (window.navigator.geolocation) {
  // 地理定位操作
} else {
  // 浏览器本机不支持地理定位
}
```

这段代码测试了 `geolocation` 属性的实现是否存在。如果存在，则执行某些地理定位操作，否则就尝试执行其他操作。

`Geolocation` 对象有三个公有方法，参见表 3-2。通过使用这几个方法和作为它们参数的回调函数，可以实现任何地理定位功能。

表3-2　Geolocation对象的方法

方　法	说　明
`clearWatch(watchId)`	停止监视与传入的 `watchId` 相关进程
`getCurrentPosition(successCallback, [errorCallback, [options]])`	尝试取得地理定位信息，成功则调用 `successCallback`，失败则调用 `errorCallback`（可选）
`watchPosition(successCallback, [errorCallback, [options]])`	尝试以固定的时间间隔取得地理定位信息，成功则调用 `successCallback`，失败则调用 `errorCallback`（可选）

3.4 取得用户位置

在验证了浏览器支持 W3C Geolocation API 之后，就可以发送请求来取得当前设备所在的位置了。为此，需要使用 `getCurrentPosition()` 方法。`getCurrentPosition()` 方法要求至少传入一个参数，即定位回调函数；还可以接受一个可选的错误回调函数和一个可选的选项参数。传入三个参数时的调用代码如下所示：

```
navigator.geolocation.getCurrentPosition(successCallback,
    errorCallback,options);
```

第一个参数 `successCallback` 会在该 API 经过内部处理成功找到位置信息后被调用。第二个参数 `errorCallback` 是可选的，会在获取定位信息出错时被调用。最后一个参数 `options` 是一个 `PositionOptions` 对象，也是可选的。



`getCurrentPosition()` 方法的 `successCallback` 参数是必需的。如果省略这个参数，那么对 `getCurrentPosition()` 方法的调用将自动失败并终止获取位置信息的进程。

以下代码片段更完整地展示调用 `getCurrentPosition()` 方法的过程：

```
if (window.navigator.geolocation) {
  navigator.geolocation.getCurrentPosition(successCallback,
      errorCallback, options);
} else {
  alert('Your browser does not natively support geolocation.');
}

function successCallback(position) {
  // 使用位置信息做些什么
}

function errorCallback(error) {
  // 取得位置信息时遇到了问题
}
```

3.4.1 PositionOptions

`PositionOptions` 对象是一个可选的参数，可以将其传递到 `getCurrentPosition()` 方法，就和 3.5 节将会介绍的一样，这个对象也是 `watchPosition()` 方法的一个可选参数。表 3-3 列出了 `PositionOptions` 对象的所有属性，这些属性也是可选的。

表3-3 **PositionOptions**对象的属性

方　法	JavaScript类型	说　明
enableHighAccurary	Boolean	标志，告诉 API 尽可能取得与设备实际位置最接近的位置信息。默认值为 false。如果把这个属性设置为 true，有可能延长响应时间或者增加电量消耗
maximumAge	Integer	表示应用可以接受缓存的位置信息，但缓存的时间不能超过指定的毫秒数。默认值为 0
timeout	Integer	表示应用从一次调用开始到 successCallback 函数被调用最长会等待多长时间，以毫秒表示。默认值为 0

下面是一个调用 getCurrentPosition() 并传入 PositionOptions 对象的例子：

```
var options = {
  enableHighAccuracy: true,
  maximumAge: 60000,
  timeout: 45000
};

navigator.geolocation.getCurrentPosition(successCallback,
    errorCallback, options);
```

以上代码调用了 getCurrentPosition()，要求返回高精确度的结果，位置信息的缓存时间不能超过 60 秒，请求时间超过 45 秒则返回错误。

3.4.2　缓存的位置信息

所谓缓存的位置信息，是指应用在过去某一时刻获取的位置信息，通过重用该信息可以不必再次请求新位置。在不需要应用频繁显示位置变化的情况下，使用缓存的位置信息是一个很好的选择。由于获取缓存的位置信息不需要再次调用 API，因而会节省一部分耗费。要指定可以接受的缓存时间，需要向 getCurrentPosition() 或 watchPosition() 方法传入 PositionOptions 对象，并将可选的 maximumAge 属性设置成期待的毫秒数。

比如：

```
var options = {
  maximumAge: 600000
};

navigator.geolocation.getCurrentPosition(successCallback,
    errorCallback, options);
```

以上代码表示应用能接受缓存时间不超过 60 分钟的位置信息。如果你每次都想得到最新的位置信息，就不要设置 maximumAge 属性（其默认值为 0），或者显式地将其设置为 0。如果你每次都想得到缓存的位置信息，那么在调用相应方法时，将这个属性设置为 Infinity，如下所示：

```
var options = {
  maximumAge: Infinity
};

navigator.geolocation.getCurrentPosition(successCallback,
    errorCallback,options);
```

3.5 更新用户位置

有的时候，应用需要随着设备改变位置而对其重新定位。在这种情况下，可以使用 watchPosition() 方法代替 getCurrentPosition() 方法。watchPosition() 方法与 getCurrentPosition() 方法的基本结构相同，它也接受一个必需的参数 successCallback 和两个可选参数 errorCallback 和 options。

这两个方法的主要区别在于，watchPosition() 方法被调用的时候会返回一个值，这个值用来唯一地标识该监视（watch）操作。监视操作本身是异步操作。以下是调用该方法的示例：

```
var watcher = navigator.geolocation.watchPosition(successCallback,
                errorCallback, options);
```

第一个参数 successCallback 会在 API 成功取得位置信息时被调用。第二个参数 errorCallback 是可选的，在取得位置信息出错时会被调用。最后一个参数 options 是一个 PositionOptions 对象，也是可选的。变量 watcher 是这次特定的监视操作的唯一标识符。

以下代码片段展示了如何使用 watchPosition() 方法：

```
var watcher = null;
var options = {
  enableHighAccuracy: true,
  timeout: 45000
};

if (window.navigator.geolocation) {
  watcher = navigator.geolocation.watchPosition(successCallback,
              errorCallback, options);
} else {
  alert('Your browser does not natively support geolocation.');
}
```

```
function successCallback(position) {
  // 使用位置信息做些什么
}

function errorCallback(error) {
  // 取得位置信息时遇到了问题
}
```

3.5.1 不需要轮询

`watchPosition()` 方法内置有自动轮询功能，可以检测设备的位置是否改变，每次轮询得到设备的新位置，都会调用 `successCallback()` 函数。这样，开发人员就不必自己编写代码每过多少秒去轮询一次设备了。由于 `watchPosition()` 方法内置了自动轮询功能，这个方法也是真正实时地理定位应用的基础。创建自定义的轮询功能最多只能给你一种伪实时的更新，而且还会额外消耗资源，造成长时间运行的应用性能降低。

只要可能，都应该使用 `watchPosition()` 方法的自动轮询能力来更新位置。除非你的应用特别需要，否则都不要自己编写代码实现轮询定位。

3.5.2 清除监视操作

与 JavaScript 中的 `clearTimeout()` 和 `clearInterval()` 方法类似，W3C Geolocation API 也提供了 `clearWatch()` 方法，调用它并传入想要清除的 `watchId`，就可以清除相应的监视操作。这个方法的语法如下：

```
navigator.geolocation.clearWatch(watcher);
```

下列代码展示了如何创建一次新的监视操作，然后在成功取得位置信息后撤销该监视：

```
var watcher = null;
var options = {
  enableHighAccuracy: true,
  timeout: 45000
};

if (window.navigator.geolocation) {
  watcher = navigator.geolocation.watchPosition(successCallback,
            errorCallback, options);
} else {
  alert('Your browser does not natively support geolocation.');
}
```

```
function successCallback(position) {
  navigator.geolocation.clearWatch(watcher);
  // 使用位置信息做些什么
}
```

3.6 处理成功的请求

当 API 成功取得了位置信息之后，就会调用 sucessCallback 函数。这种情况无论使用 getCurrentPosition() 还是 watchPosition() 方法都是一样的。在调用 sucessCallback 函数时，API 会将一个 Position 对象传给它。

3.6.1 Position对象

Position 对象中保存着 W3C Geolocation API 调用返回的所有地理定位信息，这个对象将被传递给 sucessCallback 函数。表 3-4 列出了这个对象目前的所有属性。在 API 将来的某个版本中，还有可能给这个对象增加新属性，比如地理编码（geocoding）等。

表3-4 Position对象的属性

属 性	说 明
coords	Coordinates 对象，包含地理坐标和其他属性
timestamp	DOMTimeStamp 对象，保存着获取 Position 对象的时间

3.6.2 Coordinates对象

通过 API 取得主要地理信息都保存在一个 Coordinates 对象中，这个对象是 Position 对象（参见 3.6.1 节）的属性。Coordinates 对象中保存的地理信息基于世界大地坐标系 WGS 84（参见 2.2.2 节）。目前，W3C Geolocation API 还不支持其他坐标系。表 3-5 列出了 Coordinates 对象的属性。

表3-5 Coordinates对象的属性

属 性	说 明
latitude	设备的地理坐标纬度，以十进制度数表示
longitude	设备的地理坐标经度，以十进制度数表示
altitude	设备的地理海拔高度，以 WGS 84 椭球体以上的米数表示
accuracy	经、纬度坐标的精度，以米表示
altitudeAccuracy	海拔高度的精度，以米表示。此属性值为 null，表示未被支持
heading	设备移动的方向，以从 0°～ 360°之间的度数表示。此属性值为 NaN，表示设备未移动；值为 null，表示未被支持
speed	设备当前的地面移动速度，以米每秒表示。此属性值为 null，表示未被支持

程序示例 3-1 展示了到目前为止我们介绍的所有 Geolocation 对象的方法和属性的用法。

程序示例 3-1　第一个地理定位的例子

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>A First Geolocation Example</title>
    <meta name="viewport" content="initial-scale=1.0, user-
        scalable=no"/>
    <meta charset="utf-8"/>'
    <script type="text/javascript">
      var options = {
        enableHighAccuracy: true,
        maximumAge: 1000,
        timeout: 45000
    };

    if (window.navigator.geolocation) {
      navigator.geolocation.getCurrentPosition(successCallback,
          errorCallback, options);
    } else {
      alert('Your browser does not natively support geolocation.');
    }
      function successCallback(position) {
        var output = '';

        output += "Your position has been located.\n\n";
        output += 'Latitude: ' + position.coords.latitude + "°\n";
        output += 'Longitude: ' + position.coords.longitude + "°\n";
        output += 'Accuracy: ' + position.coords.accuracy + " meters\n";
        if (position.coords.altitude)
          output += 'Altitude: ' + position.coords.altitude + " meters\n";
        if (position.coords.altitudeAccuracy)
          output += 'Altitude Accuracy: ' + position.coords.altitudeAccuracy +
              "meters\n";
        if (position.coords.heading)
          output += 'Heading: ' + position.coords.Heading + "°\n";
        if (position.coords.speed)
          output += 'Speed: ' + position.coords.Speed + " m/s\n";
        output += 'Time of Position: ' + position.timestamp;

        alert(output);'
      }
      function errorCallback(error) {
        // 取得位置信息时遇到了问题
      }
    </script>
  </head>
  <body>
    <div>A First Geolocation Example</div>
  </body>
</html>
```

这个例子虽然只使用 alert 简单地向用户显示消息，但它却展示了从 Position 对象中取得各种地理信息的方式。在真实的应用中，可以利用坐标在地图上画一个点，也可以将坐标保存到数据库中。

以编程方式转换坐标

W3C Geolocation API 返回的经、纬度坐标是十进制度数的格式。而有时候需要把这些度数转换成度分秒格式，此时要是可以不必手工转换就最好不过了。程序示例 3-2 提供了两个对象，可以完成从十进制度数到度分秒（dd2dms）及相反的转换（dms2dd）。

程序示例 3-2　十进制度数与度分秒的相互转换

```
function dd2dms(degree, lat_long) {
  this.deg = Math.abs(parseInt(degree));
  this.min = (Math.abs(degree) - this.deg) * 60;
  this.sec = this.min;
  this.min = Math.abs(parseInt(this.min));
  this.sec = Math.round((this.sec - this.min) * 60 * 1000000) / 1000000;
  this.sign = (degree < 0) ? -1 : 1;
  this.dir = (lat_long == 'lat') ? ((this.sign > 0) ? 'N' : 'S') :
  ((this.sign > 0) ? 'E' : 'W');
this.toString = function(dir) {
  if (isNaN(dir))
    return (this.deg * this.sign) + "\u00b0" + this.min + "'" +
        this.sec + '"';
    else
      return this.deg + "\u00b0 " + this.min + "'" + this.sec + '"' +
          this.dir;
  };
}

function dms2dd(deg, min, sec, dir) {
  if (dir) {
    this.sign = (dir.toLowerCase() == 'w' || dir.toLowerCase() == 's') ? -1
: 1;
    this.dir = (dir.toLowerCase() == 'w' || dir.toLowerCase() == 's'
||
        dir.toLowerCase() == 'n' || dir.toLowerCase() == 'e') ?
        dir.toUpperCase() : '';
  } else {
    this.sign = (deg < 0) ? -1 : 1;
    this.dir = '';
  }
  this.dec = Math.round((Math.abs(deg) + ((min * 60) + sec) / 3600) *
      1000000) / 1000000;
  this.toString = function(dir) {
    if (isNaN(dir) || this.dir == '')
      return (this.dec * this.sign) + "\u00b0";
    else
      return this.dec + "\u00b0" + ' ' + this.dir;
  }
}
```

这两个对象很简单，经过修改和优化还能提高其运行效率，不过对于了解如何编写转换代码已经足够了。下面的代码演示了如何使用这两个对象来完成实际的转换。

```
alert(new dd2dms(40.567534, 'long').toString(1)); // 输出：40°34'3.1224" E
alert(new dms2dd(40, 34, 3.1224, 'E').toString(1)); // 输出：40.567 534° E
```

3.7 处理请求返回的错误

有一些因素会导致通过 API 取得位置信息失败，此时只要在 getCurrentPosition() 或 watchPosition() 方法中提供了 errorCallback 函数，该函数就会被调用。在调用 errorCallback 函数时，API 会给它传入一个 PositionError 对象。

PositionError对象

PositionError 对象中保存着了 W3C Geolocation API 调用返回的所有错误信息，这个对象将被传递给 errorCallback 函数。表 3-6 列出了这个对象目前的所有属性。

表3-6 PositionError对象的属性

属性	说明
code	一个数值，表示错误类型，可以是下列值中的一个： PERMISSION_DENIED(1) 表示应用没有使用 Geolocation API 的必要权限，因而导致位置调用失败 POSITION_UNAVAILABLE(2) 表示不能确定位置导致的位置调用失败 TIMEOUT(3) 表示尝试获取位置信息所花时间超过了 timeout 属性指定的时间，因而导致位置调用失败
message	一条详细描述错误的消息，供开发人员调试使用。最好不要把这条消息显示给应用的最终用户

程序示例 3-3 展示了使用 Geolocation 对象所有 API 的代码。

程序示例 3-3　第二个地理定位的例子

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>A First Geolocation Example</title>
    <meta name="viewport" content="initial-scale=1.0, user-scalable=no"/>
    <meta charset="utf-8"/>
    <script type="text/javascript">
      var options = {
        enableHighAccuracy: true,
        maximumAge: 1000,
        timeout: 45000
```

```
        };

        if (window.navigator.geolocation) {
          navigator.geolocation.getCurrentPosition(successCallback,
              errorCallback, options);
        } else {
          alert('Your browser does not natively support geolocation.');
        }

        function successCallback(position) {
          var output = '';

          output += "Your position has been located.\n\n";
          output += 'Latitude: ' + position.coords.latitude + "°\n";
          output += 'Longitude: ' + position.coords.longitude + "°\n";
          output += 'Accuracy: ' + position.coords.accuracy + " meters\n";
          if (position.coords.altitude)
            output += 'Altitude: ' + position.coords.altitude + " meters\n";
          if (position.coords.altitudeAccuracy)
            output += 'Altitude Accuracy: ' + position.coords.altitudeAccuracy +
                " meters\n";
          if (position.coords.heading)
            output += 'Heading: ' + position.coords.Heading + "°\n";
          if (position.coords.speed)
            output += 'Speed: ' + position.coords.Speed + " m/s\n";
          output += 'Time of Position: ' + position.timestamp;

          alert(output);
        }

        function errorCallback(error) {
          switch (error.code) {
            case error.PERMISSION_DENIED:
              alert('You have denied access to your position.');
              break;
            case error.POSITION_UNAVAILABLE:
              alert('There was a problem getting your position.');
              break;
            case error.TIMEOUT:
              alert('The application has timed out attempting to get your ' +
                  'location.');
                break;
          }
        }
      </script>
    </head>
    <body>
      <div>A First Geolocation Example</div>
    </body>
  </html>
```

这个例子捕获了错误，并通过 alert 向用户显示了更容易看懂的消息。在真实的应用中，可能需要把错误消息记录到日志，或者利用这些消息为用户提供更有用的帮

助。我们这个例子的目的只是为了让大家了解如何使用通过 `PositionError` 对象传进来的错误码。

3.8 隐私问题

隐私是一个非常重要的问题，不仅 W3C Geolocation API 涉及此问题，所有地理定位应用也都涉及此问题。所有公司都知道每个人有多么重视自己的隐私，也知道应该采取什么措施保护从用户那里得到的隐私信息。作为用户，如果你不了解某公司的隐私保护策略及方式，应该查看该公司的与保护用户数据有关的隐私策略。

为了解决与 W3C Geolocation API 相关的安全及隐私问题，该规范规定在未得到用户明确授权的情况下，不得将位置信息发送到 Web 应用。所有浏览器遵循这一规定的标准实现方式，就是提供一个信息栏并按规范显示请求位置文档的 URI（参见图 3-1）。用户可以选择始终允许站点级的访问——通过选中一个复选框设置。在浏览器的设置中，用户随时可以取消这一授权。

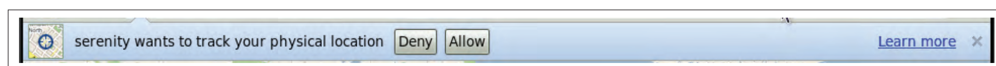


图 3-1 Chrome 浏览器的事先同意策略



从开发的角度看，如果用户不允许应用访问他的位置信息，那你应该得体地处理这件事。

用户允许应用在特定浏览器中具有站点级的访问权限后，该许可会在当前浏览器会话结束时失效。在这种事先同意策略的基础上，任何对 W3C Geolocation API 的实现都应该提供足够的隐私保护功能，尽量不要让用户为他们的位置信息而担心。作为开发人员，最关键的是要适当地使用用户数据。虽然用户通常会准许他们信任的个人或站点使用自己的数据，但也不能因此掉以轻心。

第 4 章

地理定位和地图API



第 3 章介绍了使用 W3C Geolocation API 通过 JavaScript 代码从用户的浏览器中取得位置信息。虽然那是本书的关键内容，但除非你——作为开发人员，要使用位置信息去做些什么，否则光取得位置信息用处也不大。比如说，取得用户位置信息，然后在地图上显示出该位置就是一种用途。毋庸置疑，在地图上绘制一个（或多个）位置，就是 GIS 最常见的应用。

提到 Web 地图应用，解决方案非常多，比如 Google Maps JavaScript API V3、Bing Maps AJAX Control, Version 7.0、Esri ArcGIS JavaScript API 2.2、Yahoo Maps AJAX API，还有 OpenStreetMap API v0.6，等等。这些只是一小部分而已（实际上，最多也就还有几个）。这些 API 在实现 Web 地图这个基本功能上是非常类似的。正因为如此，本章将只针对两个 API 进行讲解，至少哪个 API 对你更合适，还得由你自己去挑选。

在介绍完如何在应用中使用这些 API 之后，本章还会讨论怎样处理取得的位置信息，以便其他应用参考或将来重新绘制。这个问题的重要性丝毫不亚于在地图上绘制出刚刚取得的地理定位，因为多数 GIS 应用都需要用户的多个位置信息。本章最后，再介绍一下怎样保存地理定位信息，以便与其他应用共享。

4.1 Google地图示例

使用 Google Maps JavaScript API，可以在网页中嵌入 Google 地图。这个 API 的第 3 版经过特殊设计，速度更快，也更适合移动设备，当然也适合桌面浏览器应用。[1] 开发人员使用这个 API 很容易实现类似 http://maps.google.com/ 的嵌入式地图，而且还能定制其外观和功能，使其更加符合应用的需要。这个 API 在 Web 应用中使用得非常多，目前有 15 万个以上的站点使用它。[2]

4.1.1 Google Maps API简介

为了方便阅读，本书将 Google Maps 应用的所有代码都放在一个 HTML 文件里。在实际的应用开发中，最好是把 CSS 和 JavaScript 分别放在各自的文件里（如果应用复杂，可能会是多个文件）。

下面来看一看程序示例 4-1，其中给出了创建一个简单的 Google 地图应用的所有必备代码。

注 1：Google Maps JavaScript API V3，http://code.google.com/apis/maps/documentation/javascript/。

注 2：Mapping Success: Google Maps Case Studies，http://maps.google.com/help/maps/casestudies/。

程序示例 4-1　简单的 Google 地图

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>A Simple Google Map</title>
    <meta name="viewport" content="initial-scale=1.0, user-
       scalable=no"/>
    <meta charset="utf-8"/>
    <style type="text/css">
      html { height: 100% }
      body { height: 100%; margin: 0; padding: 0 }
      #map { height: 100% }
  </style>
  <script type="text/javascript"
      src="http://maps.google.com/maps/api/js?sensor=false"></script>
  <script type="text/javascript">
    var map;

    /* 这个函数在页面加载完毕后会立即被调用 */
    function InitMap() {
      /* 设置所有的地图选项 */
      var options = {
        zoom: 4,
        center: new google.maps.LatLng(38.6201, -90.2003),
        mapTypeId: google.maps.MapTypeId.ROADMAP,
        mapTypeControl: true,
        mapTypeControlOptions: {
          style: google.maps.MapTypeControlStyle.HORIZONTAL_BAR,
          position: google.maps.ControlPosition.BOTTOM_CENTER
        },
        panControl: true,
        panControlOptions: {
        position: google.maps.ControlPosition.TOP_RIGHT
        },
        zoomControl: true,
        zoomControlOptions: {
        style: google.maps.ZoomControlStyle.LARGE,
        position: google.maps.ControlPosition.LEFT_CENTER
      },
      scaleControl: true,
      scaleControlOptions: {
        position: google.maps.ControlPosition.BOTTOM_LEFT
      },
      streetViewControl: true,
      streetViewControlOptions: {
        position: google.maps.ControlPosition.LEFT_TOP
      }
    };
    /* 在应用中创建新地图 */
    map = new google.maps.Map(document.getElementById('map'), options);
  }

    /* 用于简化事件处理的辅助对象 */
    var Utils = { };
```

```
    Utils.addEvent = (function() {
      return function addEvent(eventObj, event, eventHandler) {
        if (eventObj.addEventListener) {
          eventObj.addEventListener(event, eventHandler, false);
        } else if (eventObj.attachEvent) {
          event = 'on' + event;
          eventObj.attachEvent(event, eventHandler);
        } else {
          eventObj['on' + event] = function() { eventHandler() };
        }
      };
    }());

    Utils.removeEvent = (function() {
      return function removeEvent(event) {
          if (event.preventDefault) {
            event.preventDefault();
            event.stopProgagation();
          } else {
            event.returnValue = false;
            event.cancelBubble = true;
          }
        };
      }());

      Utils.addEvent(window, 'load', InitMap);
    </script>
  </head>
  <body>
    <div id="map"></div>
  </body>
</html>
```

程序示例 4-1 中的代码会生成一幅类似图 4-1 所示的地图。稍后我们会逐步分析这些代码，但现在需要向大家说明几件事儿：

- 这个应用是用 HTML5 写的；
- 包含在应用中的 Google Maps JavaScript API 直接来自 Google 的站点；
- 有两个 JavaScript 函数用于辅助跨浏览器的事件处理；
- 创建 Google 地图首先要指定一个容器（`<div>` 元素）来容纳地图，然后通过一组选项来定制地图控件的外观。

在应用中指定 `DOCTYPE` 可以保证任何浏览器都会以兼容标准的方式来呈现内容。这里选择 HTML5 是因为它很快就将成为业界的标准，实际上任何 `DOCTYPE` 都是可以的。

应用中包含 Google Maps JavaScript API 的代码是下面这两行：

```
<script type="text/javascript"
   src="http://maps.google.com/maps/api/js?sensor=false"></script>
```

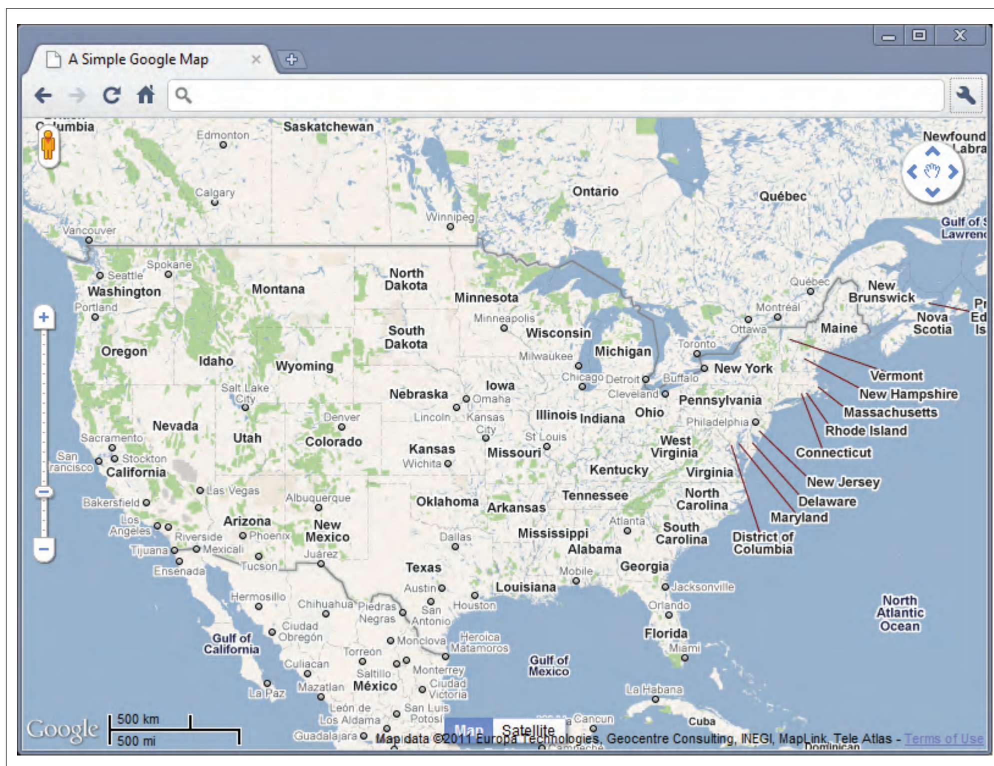


图 4-1 在 Chrome 中查看这个简单的 Google 地图应用

这样就可以保证使用该 API 的最新版本。参数 sensor 被设置为 false，表示地图不使用传感器来确定用户的位置。



在向程序示例 4-1 添加地理定位功能时，就要把 sensor 参数改为 true，以便让 API 知道位置信息来源于“传感器”，比如手机里的 GPS 定位器。

最后，在讨论与 Google Map JavaScript API 相关的代码之前，还要注意有一个 name 属性为 viewport 的 <meta> 元素，该元素目前只有 iPhone 可以识别，它的作用是将应用设置为全屏，不允许用户调整地图大小。将来还会有其他智能手机利用这个元素。

代码中的 Utils 变量就是一个对象，保存着跨浏览器的事件处理函数，以保证应用更加灵活。使用 Utils.addEvent() 方法，即可将代码添加到应用中，而且不会重写可能已经存在的 onload 函数。如果在应用中使用 jQuery 或 Dojo 等 JavaScript 库，那这些库肯定也会提供内置的方法以实现跨浏览器的事件处理。

地图，是通过初始化 `google.maps.Map` 对象的新实例来创建的，初始化时要指定容纳地图的元素。代码中的这个元素是使用 `document.getElementById()` 这个 DOM 方法来取得的。另外，初始化地图对象时还传入了一个 `options` 对象，该对象用于控制地图上的一切。

地图选项

默认情况下，Google Map JavaScript API 会提供导航地图和切换地图类型的控件。此外，默认还能够支持所有设备的键盘操作。使用地图（`map` 对象）的 `disableDefaultUI` 选项可以禁用默认的控件，而使用其他选项可以控制每一种控件。

在程序示例 4-1 中，为配置地图使用了下列选项。

`zoom`
在这个例子中，默认的缩放级别设置为 4。`zoom` 的取值范围是从 0 ～ 21+，其中 0 表示整个世界的地图，21 表示显示个别的建筑物。

`center`
使用一对纬度和经度坐标定义地图的中心点。

`mapType`
Google Map JavaScript API 支持以下地图类型：`ROADMAP`、`SATELLITE`、`HYBRID` 和 `TERRAIN`。

`mapTypeControl`、`panControl`、`zoomControl`、`scaleControl`、`streetViewControl`

使用 `true` 或 `false` 可以启用或禁用这些控件。此外，它们都有自己的配置选项，用于通过 `position` 属性设置控件的位置。

要了解有关地图选项的更多信息，请参考 Google Maps JavaScript API 的 Developer’s Guide（开发指南）和 API Reference（API 参考）。

4.1.2 向Google地图中添加地理定位

正如第 3 章所介绍的，要向程序示例 4-1 生成的 Google 地图中添加地理定位，主要涉及三个方面：调用 `getCurrentPosition()` 方法、取得位置后通过 `successCallback` 函数执行某些操作，以及在遇到问题时通过 `errorCallback` 函数处理错误。为此，我们将创建一个名为 `getLocation()` 的函数检查 `navigator.geolocation` 对象，并在该对象可用的情况下尝试获取位置信息。这个函数还可能会修改变量 `browserSupport` 的值，以便将来的 `errorCallback` 知

道错误到底来自 API，还是因为浏览器不支持地理定位。这样做主要是为了把所有错误处理逻辑放到一块，避免代码中的任何位置都可能弹出警告框。如此一来，如果我不满足于仅仅是向用户弹出警告框，而是想给出更好的错误处理方案，那么要修改也就只需修改一个地方就好了。

程序示例 4-2 展示了在 Google 地图中添加地理定位的过程，为了理解方便，其中修改或新增的代码都以粗体标出。

程序示例 4-2　向 Google 地图中添加地理定位

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Adding Geolocation to a Google Map</title>
    <meta name="viewport" content="initial-scale=1.0, user-
       scalable=no"/>
    <meta charset="utf-8"/>
    <style type="text/css">
      html { height: 100% } -
      body { height: 100%; margin: 0; padding: 0 }
      #map { height: 100% }
    </style>
    <script type="text/javascript"
        src="http://maps.google.com/maps/api/js?sensor=true"></script>
    <script type="text/javascript">
      var map;
      var browserSupport = false;
      var attempts = 0;

      /* 这个函数在页面加载完毕后会立即被调用 */
      function InitMap() {
         /* 设置所有的地图选项 */
         var options = {
           zoom: 4,
           center: new google.maps.LatLng(38.6201, -90.2003),
           mapTypeId: google.maps.MapTypeId.ROADMAP,
           mapTypeControl: true,
           mapTypeControlOptions: {
             style: google.maps.MapTypeControlStyle.HORIZONTAL_BAR,
             position: google.maps.ControlPosition.BOTTOM_CENTER
           },
           panControl: true,
           panControlOptions: {
             position: google.maps.ControlPosition.TOP_RIGHT
           },
           zoomControl: true,
           zoomControlOptions: {
             style: google.maps.ZoomControlStyle.LARGE,
             position: google.maps.ControlPosition.LEFT_CENTER
           },
           scaleControl: true,
           scaleControlOptions: {
```

```
      position: google.maps.ControlPosition.BOTTOM_LEFT
    },
    streetViewControl: true,
    streetViewControlOptions: {
      position: google.maps.ControlPosition.LEFT_TOP
    }
  };

  /* 在应用中创建新地图 */
  map = new google.maps.Map(document.getElementById('map'), options);

  /* 添加地理定位 */
  getLocation();
}

/*
 * 如果 W3C Geolocation 对象可以用就取得当前位置，否则报告问题
 */
function getLocation() {
  /* 检查浏览器是否支持 W3C Geolocation API*/
  if (navigator.geolocation) {
    browserSupport = true;
    navigator.geolocation.getCurrentPosition(plotLocation,
        reportProblem, { timeout: 45000 });
  } else
    reportProblem();
}

/* 在地图上绘制位置标记并将地图放大 */
function plotLocation(position) {
  attempts = 0;
  var point = new google.maps.LatLng(position.coords.latitude,
      position.coords.longitude);
  var marker = new google.maps.Marker({
      position: point
  });

  marker.setMap(map);
  map.setCenter(point);
  map.setZoom(15);
}

/* 通过这个函数报告错误 */
function reportProblem(e) {
  /* 判断是浏览器支持问题，还是 API 自身的问题 */
  if (browserSupport) {
    switch (e.code) {
      case e.PERMISSION_DENIED:
        alert('You have denied access to your position. You will ' +
            'not get the most out of the application now.');
          break;
      case e.POSITION_UNAVAILABLE:
          alert('There was a problem getting your position.');
          break;
        case e.TIMEOUT:
```

```
            /* 在最终判定超时之前，再给三次机会 */
            if (++attempts < 3) {
              navigator.geolocation.getCurrentPosition(plotLocation,
                  reportProblem);
              } else
                alert('The application has timed out attempting to get ' +
                    'your location.');
                break;
            default:
              alert('There was a horrible Geolocation error that has ' +
                  'not been defined.');
            }
          } else
            alert('Geolocation is not supported by your browser.');
        }

        /* 用于简化事件处理的辅助对象 */
        var Utils = { };

        Utils.addEvent = (function() {
          return function addEvent(eventObj, event, eventHandler) {
            if (eventObj.addEventListener) {
              eventObj.addEventListener(event, eventHandler, false);
            } else if (eventObj.attachEvent) {
              event = 'on' + event;
              eventObj.attachEvent(event, eventHandler);
            } else {
              eventObj['on' + event] = function() { eventHandler() };
            }
          };
        }());
        Utils.removeEvent = (function() {
          return function removeEvent(event) {
            if (event.preventDefault) {
              event.preventDefault();
              event.stopProgagation();
            } else {
              event.returnValue = false;
              event.cancelBubble = true;
            }
          };
       }());

       Utils.addEvent(window, 'load', InitMap);
     </script>
  </head>
  <body>
    <div id="map"></div>
  </body>
</html>
```

这个例子中首先要注意的是，在调用 Google JavaScript API 时，传递的参数是 sensor=true，这是因为例子中要使用地理定位。在实例化地图对象之后，随即调用了 getLocation() 函数。

然后定义了两个回调函数：`plotLocation()` 和 `reportProblem()`。`plotLocation()` 会接收到一个包含所有地理定位信息的 `Position` 对象，而 `reportProblem()` 则会接收到一个包含错误码和消息的 `PositionError` 对象。

`plotLocation()` 函数使用传入的 `Postion` 对象的纬度和经度创建了一个 `LatLng` 对象，又基于这个 `LatLng` 对象创建了 `Marker` 对象。然后将这个 `Marker` 对象放到地图上，并以当前的地理定位为中心进行放大。

`reportProlem()` 函数只负责向用户显示特定的错误消息，依据是传入的 `browserSupport` 变量或 `PositionError` 对象。如果错误是因为超时造成的，则在放弃并向用户报告问题之前，再尝试三次取得用户的当前位置。

不支持W3C Geolocation API的浏览器

程序示例 4-2 中的代码在支持 W3C Geolocation API 的浏览器中可以正常运行，但不支持该 API 的浏览器呢？也许有读者还记得，第 3 章的 3.1.2 节曾讨论过针对其他浏览器的解决方案，即 geo-location-javascript 库。这个 JavaScript 库对它提供的地理定位功能有非常严格的限制，但它提供了跨浏览器的兼容性。我们这个 Google 地图的例子非常简单，因而可以使用这个库，不必担心它有什么功能缺失。程序示例 4-3 展示了使用 geo-location-javascript 库实现的跨浏览器的解决方案。同样，修改和增加的代码也加粗了，这样看起来更方便。

程序示例 4-3　在其他浏览器中向 Google 地图上添加地理定位

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Adding Geolocation for Other Browsers to a Google Map
       </title>
    <meta name="viewport" content="initial-scale=1.0, user-scalable=
       no"/>
    <meta charset="utf-8"/>
    <style type="text/css">
      html { height: 100% }
      body { height: 100%; margin: 0; padding: 0 }
      #map { height: 100% }
    </style>
    <script type="text/javascript"
        src="http://maps.google.com/maps/api/js?sensor=true"></script>
    <script type="text/javascript" src="gears_init.js"></script>
    <script type="text/javascript" src="geo.js"></script>
    <script type="text/javascript">
      var map;
      var browserSupport = false;

      /* 这个函数在页面加载完毕后会立即被调用 */
```

```
function InitMap() {
  /* 设置所有的地图选项 */
  var options = {
    zoom: 4,
    center: new google.maps.LatLng(38.6201, -90.2003),
    mapTypeId: google.maps.MapTypeId.ROADMAP,
    mapTypeControl: true,
    mapTypeControlOptions: {
      style: google.maps.MapTypeControlStyle.HORIZONTAL_BAR,
      position: google.maps.ControlPosition.BOTTOM_CENTER
    },
    panControl: true,
    panControlOptions: {
      position: google.maps.ControlPosition.TOP_RIGHT
    },
    zoomControl: true,
    zoomControlOptions: {
      style: google.maps.ZoomControlStyle.LARGE,
      position: google.maps.ControlPosition.LEFT_CENTER
    },
    scaleControl: true,
    scaleControlOptions: {
      position: google.maps.ControlPosition.BOTTOM_LEFT
    },
    streetViewControl: true,
    streetViewControlOptions: {
      position: google.maps.ControlPosition.LEFT_TOP
    }
  };
  /* 在应用中创建新地图 */
  map = new google.maps.Map(document.getElementById('map'), options);

  /* 添加地理定位 */
  getLocation();
}

/*
 * 浏览器将使用对它而言可用的 API。但愿会使用 W3C Geolocation API
 * 取得当前位置。如果浏览器完全不支持地理定位，则向用户报告问题
 */
function getLocation() {
  /* 检查浏览器是否支持某种地理定位 API*/
  if (geo_position_js.init()) {
    browserSupport = true;
    geo_position_js.getCurrentPosition(plotLocation,
        reportProblem);
    } else
      reportProblem();
    }

    /* 在地图上绘制位置标记并将地图放大 */
    function plotLocation(position) {
      var point = new google.maps.LatLng(position.coords.latitude,
```

```
            position.coords.longitude);
          var marker = new google.maps.Marker({
              position: point
          });

          marker.setMap(map);
          map.setCenter(point);
          map.setZoom(15);
        }

        /* 通过这个函数报告错误 */
        function reportProblem() {
          /* 判断是浏览器支持问题，还是 API 自身的问题 */
          if (browserSupport)
            alert('Could not locate your device.');
          else
            alert('Geolocation is not supported by your browser.');
          }

      /* 用于简化事件处理的辅助对象 */
      var Utils = { };

        Utils.addEvent = (function() {
          return function addEvent(eventObj, event, eventHandler) {
            if (eventObj.addEventListener) {
              eventObj.addEventListener(event, eventHandler, false);
            } else if (eventObj.attachEvent) {
              event = 'on' + event;
              eventObj.attachEvent(event, eventHandler);
            } else {
              eventObj['on' + event] = function() { eventHandler() };
            }
          };
        }());

        Utils.removeEvent = (function() {
          return function removeEvent(event) {
            if (event.preventDefault) {
              event.preventDefault();
              event.stopProgagation();
            } else {
              event.returnValue = false;
              event.cancelBubble = true;
            }
          };
        }());

        Utils.addEvent(window, 'load', InitMap);
      </script>
    </head>
   <body>
   <div id="map"></div>
  </body>
</html>
```

这个例子为了取得地理定位调用了 gears_init.js 和 geo.js 这两个库。其他地图相关的功能与程序示例 4-2 完全相同。

但这个例子没有检查 `navigation.geolocation` 对象，而是初始化了 geo-location-javascript 的 API，结果会返回浏览器是否支持 geo-location-javascript 包装的地理定位 API。如果支持，则调用 `getCurrentPosition()` 方法，但没有设置超时，除此之外，`getLocation()` 与程序示例 4-2 中的同名函数非常相似。

两个添加地理定位的例子中的 `plotLocation()` 函数没有任何变化，但 `reportProblem()` 函数的变化却非常大。首先，没有给这个函数传入 `PositionError` 对象——geo-location-javascript 没有这个功能。其次，错误处理非常简单——这也这个 API 的最大的不足。如前所述，因为这个例子十分简单，所以它能正常运行。但假如地理定位的需求变复杂了，那么就不可避免地要多编写很多代码才行。

4.2 ArcGIS JavaScript API的例子

Esri 提供了基于 JavaScript 的 ArcGIS API，让开发人员能够使用地图、编辑、地理编码、地理处理等 Esri 提供的各种服务。使用这个 API，可以在应用中嵌入功能如 http://www.esri.com/ 中显示的一样的地图，而且可以对其进行定制，以使其满足应用的需要。这个 JavaScript API 放在 ArcGIS Online 上，公众可以免费使用。很多站点使用这个 API 来实现其 GIS 需求，其中使用 Esri 企业级软件的桌面及服务器 GIS 应用使用这个 API 的就更多了。在本书写作时，这个 API 的最新版本是 2.2。

4.2.1 ArcGIS JavaScript API简介

同样，为了读者查阅方便，本书将 ArcGIS JavaScript 地图应用的示例代码都放在了一个 HTML 文件中。在开发实际应用的时候，应该把 JavaScript 和 CSS 分别放在各自的文件里。

下面来看一看程序示例 4-4，其中包含创建一个简单的 ArcGIS JavaScript 地图应用必需的所有代码。

程序示例 4-4　简单的 Esri ArcGIS 地图

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=7"/>
    <meta http-equiv="viewport"
        content="initial-scale=1, maximum-scale=1, user-scalable=no"/>
```

```
<title>A Simple Esri ArcGIS Map</title>

<link rel="stylesheet" href="http://serverapi.arcgisonline.com/jsapi/
   arcgis/ \2.2/js/dojo/dijit/themes/claro/claro.css"/>
<style type="text/css">
  html, body {
    height: 100%;
    margin: 0;
    padding: 0;
    width: 100%;
  }

  #map {
    height: 100%;
    width: 100%;
  }
</style>

<script type="text/javascript">
  var djConfig = { parseOnLoad: true };
</script>
<script type="text/javascript"
    src="http://serverapi.arcgisonline.com/jsapi/arcgis/?v=2.2">
</script>
<script type="text/javascript">
  dojo.require('esri.map');

var map;
var initialExtent = {
  xmin: -119.3324,
  ymin: 26.3156,
    xmax: -72.3568,
    ymax: 55.0558,
    /*
     * Web Mercator（102113）或 WGS 84（4326）
     * 是仅有的两个支持跨日界线连接平移的参照系
     */
    spatialReference: { wkid: 4326 }
  };
  var startExtent;
  var basemap;

  function initApp() {
    var startExtent = new esri.geometry.Extent(initialExtent);

    map = new esri.Map('map', {
      extent: startExtent,
      wrapAround180: true
    });

    basemap = new esri.layers.ArcGISTiledMapServiceLayer(
        'http://server.arcgisonline.com/ArcGIS/rest/services/' +
        'ESRI_StreetMap_World_2D/MapServer');
    map.addLayer(basemap);
```

```
        }

        dojo.addOnLoad(initApp);
      </script>
    </head>
    <body class="claro">
      <div id="map"></div>
    </body>
  </html>
```

程序示例 4-4 中的代码会生成如图 4-2 所示的地图。稍后我们会逐步分析这些代码，但现在需要向大家说明几点：

- 这个应用是用 HTML5 写的；
- 包含在应用中的 Esri ArcGIS JavaScript API 来自 ArcGIS Online；
- 功能极其强大的 Dojo Toolkit 库也随同对 ArcGIS Online 的调用包含在了 API 中；
- 创建 ArcGIS 地图首先要指定一个容器（`<div>` 元素）来容纳地图，然后通过一组选项来定制地图控件的外观。

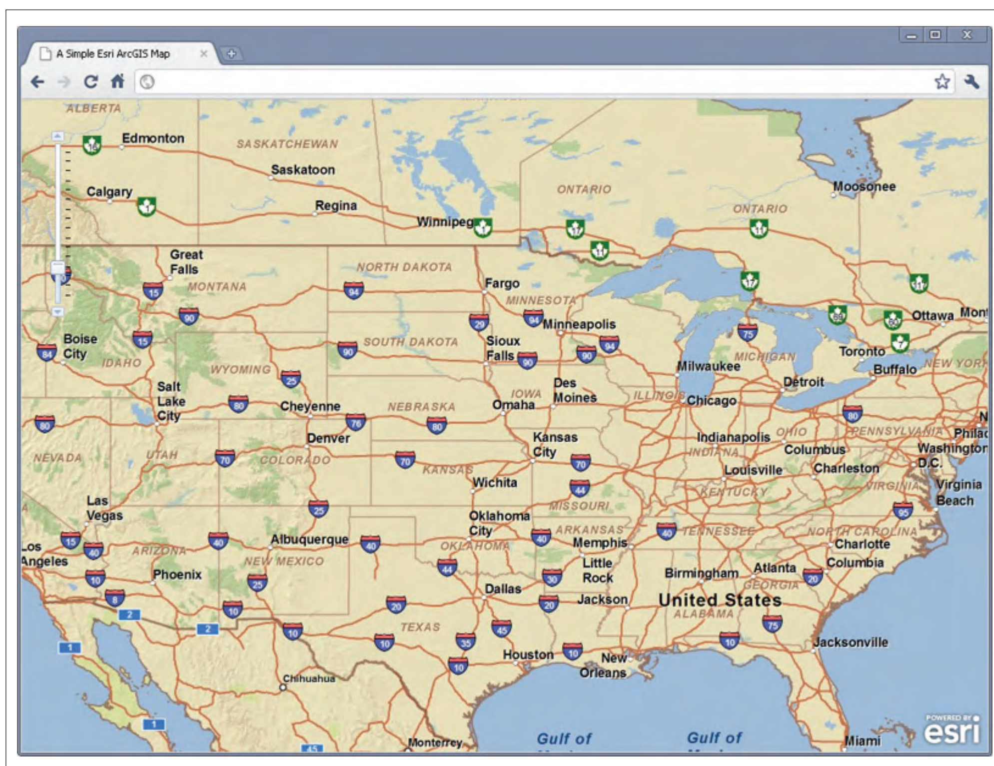


图 4-2　在 Chrome 中查看这个简单的 Esri ArcGIS 地图应用

与 Google 地图的例子一样，这里选择 HTML5 是因为它很快就将成为业界的标准，而且使用 HTML5 也为将来在地图中添加更多功能准备了条件。在应用中指定 DOCTYPE 可以保证任何浏览器都会以兼容标准的方式来呈现内容。当然，使用任何有效的 DOCTYPE 都是可以的。

应用中包含 Esri ArcGIS JavaScript API 及 Dojo Toolkit 的代码是下面这几行：

```
<script type="text/javascript"
    src="http://serverapi.arcgisonline.com/jsapi/arcgis/?v=2.2">
</script>
```

查询字符串中指定了我们想要使用的 API 版本号，在这里指定的是当前最新版本 2.2。

同样，这里也使用了一个 name 属性为 viewport 的 <meta> 元素，该元素目前只有 iPhone 可以识别，它的作用是将应用设置为全屏，不允许用户调整地图大小。除此之外，还有另一个 <meta> 元素，针对的是 Internet Explorer，告诉该浏览器将应用像在 IE 7 中一样解释并显示。随着 IE 7 用户越来越少，这个元素将来还会变。

地图是通过实例化新的 esri.Map 对象并指定容纳地图的元素创建的。元素是通过其 id 值指定的。Map 构造函数还接受一个选项参数，该参数用于控制地图的初始范围及其他地图的值。比如，wrapAround180 属性，是 2.2 版 API 中新增的，用于告诉地图是否连续地平移过日界线。这个 JavaScript API 之前的版本都不能像其他 Web 地图应用那样平移过日界线。

要了解传入 Map 构造函数的更多选项属性，或者有关该 API 的详细介绍，请访问 ArcGIS API for JavaScript Resource 页面 http://t.cn/aBxemm[3]。

4.2.2 向Esri地图中添加地理定位

读者可能已经注意到程序示例 4-1 与程序示例 4-4 这两个地图应用的相似之处了。具体来说，使用这两个 API 创建地图的方式很相似。这两个应用本身有相似之处，而向它们添加地理定位的方式则几乎完全相同。为向 ArcGIS JavaScript 应用中添加 W3C Geolocation API 代码，要做的仍然是像在 Google 地图中所做的一样。

首先，要创建一个函数 getLocation()，检测 navigator.geolocation 对象并调用 getCurrentPosition()。这个函数同样会修改全局变量 broswerSupport，以便 errorCallback 函数知道错误到底来自 API，还是因为浏览器不支持。程序示例 4-5 展示了向 Esri ArcGIS 地图的例子中添加地理定位功能的代码。为了理解方便，其中修改或新增的代码都以粗体标出。

注 3：长 URL：resources.argis.com/zh-cn/content/arcgisserver/web-apis。（译者注）

程序示例 4-5　向 Esri ArcGIS 地图中添加地理定位

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=7"/>
    <meta http-equiv="viewport"
        content="initial-scale=1, maximum-scale=1, user-scalable=no"/>

    <title>Adding Geolocation to an Esri ArcGIS Map</title>

    <link rel="stylesheet" href="http://serverapi.arcgisonline.com/jsapi/
       arcgis/ \2.2/js/dojo/dijit/themes/claro/claro.css"/>
    <style type="text/css">
      html, body {
        height: 100%;
        margin: 0;
        padding: 0;
        width: 100%;
      }

      #map {
         height: 100%;
         width: 100%;
      }
    </style>

    <script type="text/javascript">
      var djConfig = { parseOnLoad: true };
    </script>
    <script type="text/javascript"
        src="http://serverapi.arcgisonline.com/jsapi/arcgis/?v=2.2">
    </script>
    <script type="text/javascript">
       dojo.require('esri.map');

       var map;
       var initialExtent = {
           xmin: -119.3324,
           ymin: 26.3156,
           xmax: -72.3568,
           ymax: 55.0558,
           /*
            * Web Mercator（102113）或 WGS 84（4326）
            * 是仅有的两个支持跨日界线连接平移的参照系
            */
           spatialReference: { wkid: 4326 }
         };
         var startExtent;
         var basemap;
         var browserSupport = false;
         var attempts = 0;

      function initApp() {
```

```
  var startExtent = new esri.geometry.Extent(initialExtent);

  map = new esri.Map('map', {
      extent: startExtent,
      wrapAround180: true
    });

  basemap = new esri.layers.ArcGISTiledMapServiceLayer(
          'http://server.arcgisonline.com/ArcGIS/rest/services/' +
          'ESRI_StreetMap_World_2D/MapServer');
  map.addLayer(basemap);

  /* 添加地理定位 */
  dojo.connect(map, 'onLoad', function() {
    getLocation();
  });
}

/*
 * 如果 W3C Geolocation 对象可以用
 * 就取得当前位置，否则报告问题
 */
function getLocation() {
  /* 检查浏览器是否支持某种地理定位 API */
  if (navigator.geolocation) {
    browserSupport = true;
    navigator.geolocation.getCurrentPosition(plotLocation,
        reportProblem, { timeout: 45000 });
  } else
    reportProblem();
}
/* 在地图上绘制位置标记并将地图放大 */
function plotLocation(position) {
  attempts = 0;

  var pointsLayer = new esri.layers.GraphicsLayer();

  map.addLayer(pointsLayer);

  var point = new esri.geometry.Point(position.coords.longitude,
      position.coords.latitude, new esri.SpatialReference({
        wkid: 4326
      }));
  pointsLayer.add(
    new esri.Graphic(
      point,
      new esri.symbol.SimpleMarkerSymbol().setColor(
          new dojo.Color([255, 0, 0, 0.5]))
    )
  );
  map.centerAndZoom(point, 13);
}

/* 通过这个函数报告错误 */
function reportProblem(e) {
```

```
        /* 判断是浏览器支持问题，还是 API 自身的问题 */
        if (browserSupport) {
          switch (e.code) {
            case e.PERMISSION_DENIED:
              alert('You have denied access to your position. You will ' +
                  'not get the most out of the application now.');
              break;
            case e.POSITION_UNAVAILABLE:
              alert('There was a problem getting your position.');
              break;
            case e.TIMEOUT:
              /* 在最终判定超时之前，再给三次机会 */
              if (++attempts < 3) {
                navigator.geolocation.getCurrentPosition(plotLocation,
                    reportProblem);
              } else
                alert('The application has timed out attempting to get ' +
                    'your location.');
                break;
            default:
              alert('There was a horrible Geolocation error that has ' +
                  'not been defined.');
          }
        } else
          alert('Geolocation is not supported by your browser.');
      }

      dojo.addOnLoad(initApp);
    </script>
  </head>
  <body class="claro">
    <div id="map"></div>
  </body>
</html>
```

在这个例子中，对 getLocation() 函数的调用代码在一个匿名函数里，该匿名函数会在地图的 onLoad 事件发生时被调用。然后，我们定义了两个回调函数：plotLocation() 和 reportProblem()。其中，reportProblem() 与程序示例 4-2 中的同名函数完全相同，这里就不再赘述了。但是，plotLocation() 函数的变化可就大了，毕竟不同 API 向地图中添加位置的方式不一样。

plotLocation() 函数首先创建了一个名为 pointsLayer 的 esri.layers.GraphicsLayer 对象（这个层是要插入位置点的地方），然后将这个对象添加到地图中。随后又创建了一个名为 point 的 esri.geometry.Point 对象，创建该对象时传入了 Position 对象中的坐标信息。接着又在 pointsLayer 上 point 指定的位置添加了一幅新图像，并将其指定为 esri.symbol.SimpleMarkerSymbol。最后，在当前的地理定位上居中并放大地图。

不支持W3C Geolocation API的浏览器

程序示例 4-5 中的代码在支持 W3C Geolocation API 的浏览器中可以实现地理定位。对于其他浏览器，我们仍然要利用 geo-location-javascript 库重新编写代码，以便实现跨浏览器的地理定位。程序示例 4-6 展示了使用 geo-location-javascript 库在 Esri ArcGIS JavaScript API 基础上实现的跨浏览器的解决方案。同样，修改和增加的代码也加粗了，这样看起来更方便。

程序示例 4-6 在其他浏览器中向 Esri ArcGIS 地图上添加地理定位

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=7"/>
    <meta http-equiv="viewport"
        content="initial-scale=1, maximum-scale=1, user-scalable=no"/>

    <title>Adding Geolocation for Other Browsers to an Esri Map</title>

    <link rel="stylesheet" href="http://serverapi.arcgisonline.com/jsapi/
       arcgis/ \2.2/js/dojo/dijit/themes/claro/claro.css"/>
    <style type="text/css">
      html, body {
        height: 100%;
        margin: 0;
        padding: 0;
        width: 100%;
      }

      #map {
      height: 100%;
      width: 100%;
    }
  </style>

  <script type="text/javascript">
    var djConfig = { parseOnLoad: true };
  </script>
  <script type="text/javascript"
      src="http://serverapi.arcgisonline.com/jsapi/arcgis/?v=2.2">
  </script>
  <script type="text/javascript" src="gears_init.js"></script>
  <script type="text/javascript" src="geo.js"></script>
  <script type="text/javascript">
    dojo.require('esri.map');

    var map;
    var initialExtent = {
        xmin: -119.3324,
        ymin: 26.3156,
        xmax: -72.3568,
        ymax: 55.0558,
```

```
    /*
     * Web Mercator（102113）或 WGS 84（4326）
     * 是仅有的两个支持跨日界线连接平移的参照系
     */
    spatialReference: { wkid: 4326 }
};
var startExtent;
var basemap;
var browserSupport = false;

function initApp() {
  var startExtent = new esri.geometry.Extent(initialExtent);

  map = new esri.Map('map', {
      extent: startExtent,
      wrapAround180: true
    });

  basemap = new esri.layers.ArcGISTiledMapServiceLayer(
          'http://server.arcgisonline.com/ArcGIS/rest/services/' +
          'ESRI_StreetMap_World_2D/MapServer');
  map.addLayer(basemap);

  /* 添加地理定位 */
  dojo.connect(map, 'onLoad', function() {
    getLocation();
  });
}

/*
 * 浏览器将使用对它而言可用的 API。但愿会使用 W3C Geolocation API
 * 取得当前位置。如果浏览器不支持地理定位，则向用户报告问题
 */
function getLocation() {
  /* 检查浏览器是否支持某种地理定位 API */
  if (geo_position_js.init()) {
    browserSupport = true;
    geo_position_js.getCurrentPosition(plotLocation,
        reportProblem);
  } else
    reportProblem();
}

/* 在地图上绘制位置标记并将地图放大 */
function plotLocation(position) {
  attempts = 0;

  var pointsLayer = new esri.layers.GraphicsLayer();

  map.addLayer(pointsLayer);

  var point = new esri.geometry.Point(position.coords.longitude,
      position.coords.latitude, new esri.SpatialReference({
        wkid: 4326
      }));
```

```
        pointsLayer.add(
          new esri.Graphic(
            point,
            new esri.symbol.SimpleMarkerSymbol().setColor(
                new dojo.Color([255, 0, 0, 0.5]))
          )
        );
        map.centerAndZoom(point, 13);
      }

      /* 通过这个函数报告错误 */
      function reportProblem() {
        /* 判断是浏览器支持问题，还是 API 自身的问题 */
        if (browserSupport)
          alert('Could not locate your device.');
        else
          alert('Geolocation is not supported by your browser.');
        }

        dojo.addOnLoad(initApp);
      </script>
    </head>
    <body class="claro">
      <div id="map"></div>
    </body>
  </html>
```

这个例子为了取得地理定位调用了 gears_init.js 和 geo.js 这两个库。其他地图相关的功能与程序示例 4-5 完全相同。

但这个例子没有检查 `navigation.geolocation` 对象，而是调用了 geo-location-javascript 的 `init()` 函数，结果会返回浏览器是否支持 geo-location-javascript 包装的地理定位 API。如果支持，则调用 `getCurrentPosition()` 方法，但没有设置超时，除此之外，`getLocation()` 与程序示例 4-5 中的同名函数非常相似。

两个 Esri 地图的例子中的 `plotLocation()` 函数没有任何变化，但 `reportProblem()` 函数的变化却非常大。主要是没有给这个函数传入 `PositionError` 对象，因为 geo-location-javascript 没有这个功能。同样的情况在程序示例 4-3 中也出现过。

如果想实现更加复杂的跨浏览器地理定位功能，则需要编写更多更复杂的代码。但愿 geo-location-javascript 以后还能在其核心代码中添加更多功能，更好地模仿 W3C Geolocation API 的方法和属性。但在这一天到来之前，相关的功能还需要开发人员自己动手来实现。可能还有一种情况，那就是所有人都抛弃过时的浏览器和手机——但不幸的是，几年来我一直没有看到这种情况发生。

第 5 章

保存地理信息



除了在地图上显示地理定位之外，很多应用还会实现其他更多功能。比如说，把位置信息保存起来以备将来使用——可能是显示用户过去曾到过什么地方，或者显示某个时间多个用户都在哪里。在这种情况下，浏览器中的应用需要取得设备的地理定位，然后将其发送到服务器以备将来处理。有关后台数据处理的内容绝大多数都超出了本书的范畴，但 Web 服务器端的语言则不过是 PHP、Python、C# 或 VB.NET、Java，等等。使用哪种语言都不重要，重要的是如何保存信息。



要了解服务器端脚本语言的更多内容，可以参考一些入门的书籍。比如，Luke Welling 和 Laura Thomson 合著的 *PHP and MySQL Development, 4th Edition*（Addison-Wesley Professional）、Mark Lutz 的 *Programming Python, Fourth Edition*（O'Reilly Media）、Kathy Sierra 和 Bert Bates 合著的 *Head First Java*，*Second Edition*（O'Reilly Media）以及 Imar Spaanjaars 的 *Beginning ASP.NET 4: in C# and VB*（Wrox）。

本章将主要关注浏览器取得地理信息之后怎么办，而不是关注如何在服务器上操作这些信息。因此，随后几节更多地会讨论规范而非实现。要想把地理信息保存起来备用有很多方式可以选择：文本文件、CSV 文件、XML 文件、JSON 文件、KML 文件、Shapesfile、Geodatabase、关系型数据库，等等。选择哪种方式保存这些几何图形信息，取决于 GIS 环境、操作系统以及预算等因素。

例如，如果你的预算有限，根据 GIS 的需要选择一种开源方案恐怕更合适。在这种情况下，使用 KML 及 Google Maps 就是正确的选择。如果你是在开发企业方案，那么使用 ArcGIS Desktop 及其他 Esri 产品的可能性更大。在开发企业方案时，可能还需要一个非常稳定的 Oracle 数据库。总而言之，一个问题有很多种不同的解决方案——知道这一点，对在自己的项目中合理取舍是很重要的。

本书只讨论三种保存数据的方式：KML、Shapefile 和关系型数据库，因为这都是用于保存地理信息的最普遍方式。万一这几种方式都不符合你的需要，至少还能让你了解使用不同的格式来保存数据的知识。

5.1 KML

KML（Keyhole Markup Language，Keyhole 标记语言）是一种 XML 格式的语言，专门用于保存地理信息。Google Maps 及 Google Earth 等基于 Internet 的浏览器或地图应用使用 KML 文件能够实现数据的可视化。KML 格式最初由 Keyhold 公司开发，该公司于 2004 年被 Google 收购。Google 将 KML 2.2 提交给了 OGC（Open Geospatial Consortium，开放地理空间联盟），以确保 KML 依旧是一个开放的标准。

2008 年 4 月 14 日，KML 成为 OGC 的官方标准。



我们经常会看到以 KMZ 为扩展名的文件，这些文件是一或多个 KML 文件及其相关图标、图像文件的压缩格式。

对于地理空间信息而言，KML 有多种用途。其中之一就是保存点地标数据——相信读者朋友肯定已经心中有数了，这正是地理定位的核心所在。在 KML 中，点地标保存在一个 `<Placemark>` 容器中。这个容器里至少包含 `name`、`description` 和 `Point` 等信息。以下 KML 文件可以表示一个简单点地标：

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
  <Placemark>
    <name>Simple placemark</name>
    <description>This is an example of a simple placemark.</description>
    <Point>
      <coordinates>-90.185278,38.624722</coordinates>
    </Point>
  </Placemark>
</kml>
```

有三种基本的点地标：

- 简单点地标
- 浮动点地标
- 凸出点地标

简单点地标只会被添加到地面上，也就是只会被显示在底层区域的高度范围内。浮动点地标则具有特定高度，而且在地面的高度之上。从具有特定高度这个角度看，突出点地标与浮动点地标类似，但却有一个自定义的“绳索”将它“拴”到地面上。这三种点地标都通过放在 `<Placemark>` 容器内的 `<Point>` 元素中的数据来控制。

下面这段代码示范了点地标的语法，展示了与地理定位有关的子元素。要了解 `<Placemark>` 可以包含的所有元素，请参考 KML 参考的 Placemark 部分，网址为 http://code.google.com/intl/zh-CN/apis/kml/documentation/kmlreference.html#placemark。

```
<Placemark id="ID">
  <name>...</name>                         <!-- string -->
  <description>...</description>           <!-- string -->
  <Timestamp>
    <when>...</when>                       <!-- kml:dateTime -->
  </Timestamp>
  <ExtendedData>...</ExtendedData>         <!-- custom -->
  <Point id="ID">
    <extrude>...</extrude>                 <!-- boolean -->
```

```
    <altitudeMode>...</altitudeMode>
            <!-- clampToGround, relativeToGround, or absolute -->
    <coordinates>...</coordinates>     <!-- long,lat[,alt] -->
  </Point>
</Placemark>
```

看一看 `<Point>` 元素，它包含三个子元素：`<extrude>`、`<altitudeMode>` 和 `<coordinates>`。其中 `<coordinates>` 是上述三种点地标都必需的一个元素，它包含以 WGS 84 为参照系、以十进制度数表示的经度和纬度，以及一个可选的以米为单位表示的海平面以上高度。在 `<Point>` 元素中有 `<altitudeMode>` 元素的情况下，点地标有可能是浮动点地标，也有可能是凸出点地标。到底是哪种点地标，取决于 `<extrude>` 元素是否以数值 1 的形式将其值设置为 `true`。

程序示例 5-1 展示了包含几个点的 KML 文件，其中包含的一些信息都是能够通过 W3C Geolocation API 获取的。

程序示例 5-1　包含地理定位信息的示例 KML 文件

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
  <Document>
    <Placemark id="pt_000000">
      <name>Point 000000</name>
      <description>This is the first point collected.</description>
      <Timestamp><when>2011-04-06T23:24:12+06:00</when></Timestamp>
      <ExtendedData>
        <Data name="accuracy"><value>20</value></Data>
        <Data name="altitudeAccuracy"><value>100</value></Data>
        <Data name="heading"><value>NaN</value></Data>
        <Data name="speed"><value>0</value></Data>
      </ExtendedData>
      <Point>
        <extrude>0</extrude>
        <altitudeMode>relativeToGround</altitudeMode>
        <coordinates>-90.185278,38.624722,212</coordinates>
      </Point>
    </Placemark>
    <Placemark id="pt_000001">
      <name>Point 000001</name>
      <description>This is the second point collected.</description>
      <Timestamp><when>2011-04-07T00:15:37+06:00</when></Timestamp>
      <ExtendedData>
        <Data name="accuracy"><value>10</value></Data>
        <Data name="altitudeAccuracy"><value>10</value></Data>
        <Data name="heading"><value>37</value></Data>
        <Data name="speed"><value>15.6464</value></Data>
      </ExtendedData>
      <Point>
        <extrude>0</extrude>
        <altitudeMode>relativeToGround</altitudeMode>
```

```
        <coordinates>-89.788221,38.4233,18</coordinates>
      </Point>
    </Placemark>
    <Placemark id="pt_000002">
      <name>Point 000002</name>
      <description>This is the third point collected.</description>
      <Timestamp><when>2011-04-07T11:49:03+06:00</when></Timestamp>
      <ExtendedData>
        <Data name="accuracy"><value>60</value></Data>
        <Data name="altitudeAccuracy"><value>80</value></Data>
        <Data name="heading"><value>147</value></Data>
        <Data name="speed"><value>31.2928</value></Data>
      </ExtendedData>
      <Point>
        <extrude>0</extrude>
        <altitudeMode>relativeToGround</altitudeMode>
        <coordinates>-90.123129,37.992331,25</coordinates>
      </Point>
    </Placemark>
  </Document>
</kml>
```

虽然经度、纬度、海拔高度以及时间戳这些信息都可以直接添加，但其他地理定位信息——精度（accuracy）、海拔精度（altitudeAccuracy）、前进方向（heading）及速度（speed），也都必须添加到 `<ExtendedData>` 元素中，在那里定义备用。有三种向 `<ExtendedData>` 元素中添加数据的方式，要了解详细信息，请参考 KML 参考的 ExtendedData 部分，网址为 http://code.google.com/intl/zh-CN/apis/kml/documentation/kmlreference.html#extendeddata。程序示例 5-1 中使用的数据对的方法适合在 Google Earth 显示值，但具体到你的应用，可能其他方法更合适。

KML 基本上就是一个文本文件，因此在应用的服务器端通过编程创建、读取或写入这种文件是非常简单的，而且与使用的技术无关。加上 KML 本身也是 XML，所以将这种格式转换为其他格式也没有那么困难。操作 KML 文件的便利使其成为保存地理定位数据的一个不错的选择。

5.2 Shapefile

Shapefile 是一种专门用来保存地理矢量数据（如点和多边形）及其相关属性的数据格式。Shapefile 由 Esri 开发和维护，最初是专为 Esri 的 ArcGIS Desktop 产品开发的一种空间数据格式，现在很多其他软件也支持这种格式，比如：AutoCAD Map、MapInfo、GeoMedia 和 GRASS。

有很多工具可以用来在 Shapefile 文件与其他文件格式之间相互转换，因而 Shapefile 也是保存地理定位信息的一种灵活的格式。保存为 Shapefile 格式的点数据可以在必

要时轻易转换成其他格式，SHP2KML、shp2CAD 以及 SHP2MIF 是一些常见的转换程序。在网上稍加搜索，就可以找到能够实现相反方向转换的程序。

虽然名叫 Shapefile，但这种格式实际上要包含一组文件，这些文件在一起才能生成必要的数据。一套 Shapefile 文件通常包含三个或更多个文件，如表 5-1 所示。

表5-1　Shapefile格式相关的文件[a]

扩　展　名	说　　明	是否必需
.shp	用于存储要素几何的主文件	是
.shx	用于存储要素几何索引的索引文件	是
.dbf	用于存储要素属性信息的 dBASE 表	是
.sbn/.sbx	用于存储要素空间索引的文件	否
.fbn/.fbx	用于存储只读 shapefile 的要素空间索引的文件	否
.ain/.aih	用于存储某个表中或专题属性表中活动字段属性索引的文件	否
.atx	用于存储 dBASE 表属性索引的文件	否
.ixs	用于存储读 / 写 shapefile 的地理编码索引	否
.mxs	用于存储读 / 写 shapefile（ODB 格式）的地理编码索引	否
.prj	用于存储坐标系信息的文件	否
.xml	用于存储 shapefile 的相关信息	否
.cpg	用于存储指定用于标识 shapefile 使用的字符集的代码页	否

a ArcGIS Resource Center, Desktop 10, Shapefile文件扩展名，http://help.arcgis.com/zh-cn/arcgisdesktop/10.0/help/index.html#/na/ 005600000003000000/。

要在 Web 应用中使用 Shapefile 文件，必须以编程方式操作它（创建、读取、写入等）。Shapefile C Library 为编写 C 程序读取、写入和更新 Shapefile 文件的开发人员提供了便利。另一个在 Web 应用开发中比较有用的脚本库是 Python Shapefile Library。

Python Shapefile Library

Python Shapefile Library（PSL）的作者是 Joel Lawhead。这个库为使用 Python 脚本语言读写 Shapefile 提供了方便，尽可能涵盖了与 Shapefile 操作相关的各种功能，不仅是创建 Shapefile 文件，而且还具备一些验证功能确保生成正确的文件。下面看一看程序示例 5-2。

程序示例 5-2　使用 Python Shapefile Library 创建 Shapefile

```
# 包含 Python Shapefile Library
import shapefile as sf

# 要创建的 Shapefile 的文件名
filename = 'shapefiles/geolocation'

# 创建一个保存点的 Shapefile 文件，打开 autoBalance
```

```
sf_w = sf.Writer(sf.POINT)
sf_w.autoBalance = 1

# 添加点
sf_w.point(-90.185278, 38.624722, 212)
sf_w.point(-89.788221, 38.4233, 18)
sf_w.point(-90.123129, 37.992331, 25)

# 创建特性信息
sf_w.field('Name', 'C', 20)
sf_w.field('Description', 'C', 80)
sf_w.field('Timestamp', 'D')
sf_w.field('Accuracy', 'N', 4, 0)
sf_w.field('AltitudeAccuracy', 'N', 4, 0)
sf_w.field('Heading', 'N', 9, 6)
sf_w.field('Speed', 'N', 7, 4)

# 添加特性信息
sf_w.record('Point 000000', 'This is the first point collected.', \
    '2011-04-06T23:24:12+06:00', 20, 100, None, 0)
sf_w.record('Point 000001', 'This is the second point collected.', \
    '2011-04-07T00:15:37+06:00', 10, 10, 37, 15.6464)
sf_w.record('Point 000002', 'This is the third point collected.', \
    '2011-04-07T11:49:03+06:00', 60, 80, 147, 31.2928)

# 保存文件
sf_w.save(filename)

# 创建投影文件
prj = open("%s.prj" % filename, 'w')
epsg = 'GEOGCS["WGS 84",DATUM["WGS_1984",SPHEROID["WGS 84",6378137, \
    298.257223563]],PRIMEM["Greenwich",0],UNIT["degree", \
    0.0174532925199433]]'
prj.write(epsg)
prj.close()
```

第一行代码将 PSL 导入到脚本中。在使用 `Writer` 对象指定了要创建的 Shapefile 类型为 `POINT` 之后，将 `autoBalance` 属性设置为 `true`（值为 1）。打开这个属性可以确保通过脚本添加了一个点或一条记录之后，另一方面信息能够自动添加（每个点都有一条对应的记录，每一条记录都对应一个点）。接下来使用 `point()` 方法添加点，这个方法接受纬度、经度和可选的海拔高度作为参数。程序示例 5-2 中为每个点都指定了纬度、经度和海拔高度。

在将特性记录添加到 Shapefile 文件之前，必须使用 `field()` 方法定义特性。`field()` 方法接受字段名、字段类型、字段长度和（数值的）小数长度。定义之后，则使用 `record()` 方法为每个点创建一条记录。在添加记录之后，保存 Shapefile 文件，生成了三个必备的文件（.shp、.shx 和 .dbf）。此外，程序示例 5-2 还创建了一个 .prj 文件，让这个 Shapefile 实例更加完整。

多数情况下，当应用需要添加另一条记录时，保存地理定位信息的 Shapefile 可能早已创建好了。下面的代码是一小段编辑已有 Shapefile 文件的脚本，这个脚本向 Shapefile 文件中又添加了一个点：

```
import shapefile as sf
filename = 'shapefiles/geolocation'
sf_e = sf.Editor(shapefile = filename + '.shp')
sf_e.point(-102.125532, 34.223411, 40)
sf_e.record('Point 000004', 'This is an appended point. ', \
'2011-04-10T01:52:22+06:00', 20, 30, 118, 17.21)
sf_e.save(filename)
```

编辑已有 Shapefile 文件并向其中添加点的代码很简单。而更新一个已有的点则要复杂一些。`Editor` 对象负责在 Shapefile 文件中插入和删除记录。首先必须读取 Shapefile 文件并找到相应记录的位置（也是 PSL 可以做到的），然后使用 `delete()` 方法删除该记录，再将带有校准后信息的新记录添加到 Shapefile 中。

即便对 Python 了解不多，使用 Python Shapefile Library 也很容易。这个库的缺点是它的文档不够完备。除此之外，可以说它是 Web 应用开发中操作 Shapefile 的非常理想的工具。

5.3 数据库

数据库是一个有组织的数据集合，便于数据的存储、操作和检索。可以用来存储地理定位信息的数据库一般是关系数据库管理系统（RDBMS，Reletional Database Management System），对象数据库管理系统（ODBMS，Object Database Management System）也是可以的。本章后面提到数据库的时候，指的都是 RDMBS。常见的 RDMBS 有 dBASE、Microsoft SQL Server、MySQL、Oracle、PostgreSQL 和 Sybase。

空间数据库是为了在同一个数据库中保存空间数据及其特性而设计的。MySQL、DB2、Oracle，还有 Microsoft SQL Server（从 2008 开始）都可以在它们的表中原生存储空间信息。但在某些情况下，为了在数据库中实现某些地理功能（特别是查询），还需要在 RDBMS 之上配合一些软件。ArcSDE、OracleSpatial 和 PostGIS 就是一些配合数据库使用，以便操作地理数据的一些软件。OracleSpatial 是专门为 Oracle 设计的，而 PostGIS 是专门为 PostgreSQL 设计的，但 ArcSDE 能够与 4 大商业数据库协作。MySQL 内置了地理功能，不需要使用额外的软件。

5.3.1 SDE

ArcSDE，也简称为 SDE（Spatial Database Engine），是 Esri 为在关系数据库中存储和管理地理数据及其他业务数据而开发的一个产品。SDE 能够与一些商业数据

库 IBM DB2、Informix、Microsoft SQL Server 和 Orcale 协作，还能在开源数据库 PostgreSQL 上运行。从 ArcSDE 9.2 开始，Esri 停止单独销售 ArcSDE，将其作为 ArcGIS Desktop 和 ArcGIS Server 的一个组件打包发售。在本书写作时，这个软件的最新版本是 10.0。ArcSDE 支持很多标准，包括 OGC Simple Feature、ISO（International Organization for Standardization）空间类型、OracleSpatial 格式、PostGIS 格式以及微软的空间数据格式。

5.3.2 PostGIS

PostGIS 为关系数据库 PostgreSQL 添加了空间数据处理功能，它是由 Refractions Research 以开源项目的形式开发的，遵循 GNU General Public License。PostGIS 的第一个稳定版本（1.0）是 2005 年发布的，本书写作时的当前版本为 1.5.2。PostGIS 与 ArcSDE 或 OracleSpatial 类似，也遵循 OGC Simple Feature 规范，但并未获得 OGC 的兼容认证。

通过下面的 SQL 语句可以大致看出 PostGIS 的一些功能：

```
SELECT loc.the_geom
FROM
geolocations loc INNER JOIN
(SELECT the_geom
FROM
  (SELECT the_geom, ST_Area(the_geom) AS area
   FROM parks) p
WHERE
 area > 10000) park ON ST_Intersects(loc.the_geom, park.the_geom)
```

这条查询语句要查找城市停车场中面积大于 10 000 平方英尺[1]的所有地理定位信息。为此，它先找到停车场的多边形，使用 PostGIS 函数 `ST_Area()` 计算它的面积。然后再找到面积大于 10 000 平方英尺的停车场。最后，使用 `ST_Intersects()` 函数找到这些停车场的地理定位信息。查询返回的结果是面积大于 10 000 平方英尺的城市停车场的地理定位信息的几何体。

5.3.3 MySQL

MySQL 是全世界最流行的开源数据库，Google、Wikipedia、YouTube 和 Facebook 等流量巨大的站点都在使用它。MySQL 实现了 OGC SQL 规范 Geometry Types 的子集，使用它无需使用别的软件。自从加入空间信息处理功能起，MySQL 已经发布了多个版本，而且已经取得了与 PostGIS 和 OracleSpatial 等空间数据库类似的地位。

注 1：10 000 平方英尺 = 929.0304 平方米。（编者注）

MySQL 5.6 之前的版本都实现 OGC 命名约定。而在本书写作时，被广泛使用的社区发布版本 MySQL 5.5.11 在命名上仍然与规范有出入。例如，MySQL 5.6 可以使用 5.3.2 节示例中相同的代码。但在 MySQL 5.5 中，相应功能的代码则要写成类似如下所示：

```
SELECT loc.the_geom
FROM
geolocations loc INNER JOIN
(SELECT the_geom
FROM
 (SELECT the_geom, Area(the_geom) AS area
  FROM parks) p
WHERE
 area > 10000) park ON Intersects(loc.the_geom, park.the_geom)
```

显然，两种写法本质上很相近，任何有 SQL 和空间数据库经验的人都能够处理好 MySQL 的版本问题。等 MySQL 5.6 被广泛使用之后，MySQL 就可以追上它的竞争对象。到那时，作为一个极其流行的关系数据库，MySQL 将成为空间数据管理领域中的一颗耀眼的明星。

在有关使用关系数据库管理空间数据的讨论最后，我们再来看一个例子。程序示例 5-3 创建了用于保存与 5.1 节或 5.2 节示例中相同的地理定位信息的数据结构。

程序示例 5-3　在 MySQL 中创建地理定位数据库

```
CREATE DATABASE geolocations;

USE geolocations;

CREATE TABLE positions (
pos_id             INT                NOT NULL AUTO_INCREMENT PRIMARY KEY,
the_geom            POINT              NOT NULL,
altitude            DECIMAL(8, 2)      NOT NULL,
accuracy            DECIMAL(4, 0)      NOT NULL,
altitudeAccuracy    DECIMAL(4, 0)      NULL,
heading             DECIMAL(9, 6)      NULL,
speed               DECIMAL(7, 4)      NULL,
timestamp           DATETIME           NOT NULL,
name                VARCHAR(20)        NOT NULL,
description         VARCHAR(80)        NULL
);
```

这个例子创建了一个名为 `geolocations` 的新数据库，然后又创建了一个名为 `positions` 的表，用于保存可以通过 W3C Geolocation API 取得的特性数据。而向这个数据库中插入一条位置记录的 SQL 语句如下所示：

```
INSERT INTO positions (
the_geom,
```

```
altitude,
accuracy,
altitudeAccuracy,
heading,
speed,
timestamp,
name,
description
) VALUES (
GeomFromText('POINT(-89.788221 38.4233)'),
18,
10,
10,
37,
15.6464,
'2011-04-07 00:15:37',
'Point 000001',
'This is the second point collected.'
);
```

这条 SQL 语句使用 `GeoFromText()` 函数以 OGC 的 WKT（Well-Known Text）格式向数据库中添加了一个点。这条 SQL 语句将在服务器端脚本中执行，执行时将使用客户端取得的位置信息。数据库表的创建和插入操作与任何关系数据库几乎没有什么不同。

第 6 章

基于地理定位开发应用



毫无疑问，地理定位技术的应用在未来几年还将继续发展——只要看一看手机上日新月异的 LBS 服务就知道了。在 Foursquare、Gowalla、Twitter、Glympse 及其他应用快速发展的同时（以及它们仍将快速发展的未来），W3C Geolocation API 还会给这个领域的本地浏览器应用开发打开更多的方便之门。听听下面这些评论吧。

- 地理定位应用已经从令人新奇的"蓬头稚子"成长为今天的移动互联网上文质彬彬、引领潮流的"当红小生"。
- 手机越变越"聪明"，随着越来越多人把旧手机换成智能终端，全球智能手机市场开始升温，一片神秘的蓝海正浮出地平线。
- 多一部智能手机，就多一部 GPS 接收机，就多一部能够利用地理定位技术的设备。
- 很多公司都认识到，在日趋流行的 LBS 服务上做广告有着巨大的市场空间，关键在于这些广告可以直达每一个人和每一个具体位置。
- HTML5 和 W3C Geolocation API 支持在网站中添加地理定位功能，而且不需要过去那些特定于应用的地理定位软件。

此外，随着时间推移，位置信息的精确程度也将持续提高。对某些人而言，这会使得地理定位更加有吸引力，而对另一些人（特别是关注隐私的人）来说，这也可能会带来更多困扰。已有的取得位置信息的所有方法都会进一步提高结果的精度，或许还会有一开始就能返回高精度数据的新方法问世。由于新的、强大的基站不断增加，基于 Cell ID 进行三角测量得到的结果也会越来越准确（基站越多，可以参与计算的信号也就越多）。随着新的卫星上天，也得益于民间与政府部门更紧密的合作，GPS 技术同样也会不断改进。当然，在更多地理定位研究者加入的条件下，IP 地址跟踪技术也会更上一层楼。



成都中国电子科技大学的一位计算机科学家王勇（音），与其伊利诺斯州埃文斯顿西北大学的同事，发明了一种仅通过 IP 地址就能在 100 米的范围内（平均 690 米）定位设备的方法。这种方法不会涉及任何用户信息！

2009 年，随着 Foursquare 引领社交媒体应用的兴起，地理定位一下子成了街谈巷议的热门词汇。接着，技术作者们把 2010 年描绘成"签到年"和"地理定位年"。自 2011 年至今，这一趋势仍在延续，而且还被说成是"移动革命"。今年及之后的几年，还将有更多的商业方案与地理定位和社交媒体应用联姻。而几种移动应用形式（地理定位、社交媒体、增强现实）的混搭，再加上几个现有应用的合并，最终会使这个市场变得更加生机勃勃、百花齐放、成熟稳健。

6.1 地理营销

地理营销（geomarketing）指的是专门针对地理定位开展营销活动。Facebook 上的广告投放已经采用了地理营销策略，即根据登录到站点的用户的 IP 地址来为用户定制广告。Google、Bing 等搜索引擎同样也采取了这一策略。实践证明，这种*被动式*的地理营销方式的效果非常好，而*主动式*的地理营销则可能是广告业的未来所在。

6.1.1 特价与新品

在社交媒体与地理定位的基础上，很多公司开始在移动平台上展开了主动式的地理营销。这种地理营销的典型做法就是 Foursquare 针对在某些位置消费提供*特价优惠*。这些特价的商品或服务有时候还可以根据个人的情况进一步"特殊化"，比如对于"地主"——在某一位置签到次数最多的人，会给予专有的待遇或额外的优惠。

很多尚未具备条件开展地理营销活动的公司都已经认识到，像 Foursquare 目前这样与商业和广告联姻，势必带来巨大的客户增长。地理营销还可能增加之前没有预见到的客户忠诚度，从而能够带来更多收入。地理营销除了在社交媒体应用中有巨大的潜力，很多团购网站也要依赖于地理定位。类似 Groupon 这样的应用根据用户的地理位置，同样可以为他们定制商品或服务。

将来，地理营销还会更加依赖客户的当前位置，用于跟踪用户移动的应用是让这些公司基于位置推广的重要基础。以下情景很快就能成为可能：一个人把车停在了商店门口，短信、电子邮件或其他形式的通知一下子都出现在了他的手机上，告诉他这条街有三家商店有特价。实现这种功能的技术已经存在了。

6.1.2 众包

众包（crowdsourcing）是由两个字组合而成的：众（crowd）和包（outsourcing）。顾名思义，众包就是指集合一大群各种各样的人的集体智慧，共同完成一个原先由个人或小团队完成的任务。互联网为众包的思想成为现实提供了基础，让一大群人共同协作成为一件极其简单的事。

在众包和地理定位技术的基础上，世界各地有关自然灾害和社会冲突的地理信息也越来越多。每当这些事件发生时，人人都成了现场地理空间分析师。比如，2010 年海地发生地震、2010 年英国石油公司（British Petroleum）原油泄露、2010 年和 2011 年中东社会局势动荡，以及 2011 年日本地震引发的海啸等。英国石油公司在清理泄露的原油时，就利用 ArcGIS Mobile 进行现场监控，而这也是众包的一个现实的例子。

很多公司也逐渐意识到众包的优势，包括：

- 迅速扩展某个想法，而不会带来很大的成本；
- 一大群聪明的一起攻克某个难题；
- 很快就能知道客户对品牌及产品的期望和感觉；
- 给客户“发出声音”的机会，能够提升品牌忠诚度。

Yelp 把众包的概念引入到了社交媒体中，让用户可以针对其数据库中的任何对象发表提示（Tips）和评论（Reviews）。此后，又有很多社交媒体也实践了相同的思想。使用这些应用可以事先了解某些商家的信息，更重要的是，通过评论某些服务和商品，客户能够真正与商家达成有形的联系——他们自己也会对某个品牌或商家产生拥有感。

而市场营销部门借助用户发表的提示和评论，就可以修正自己的广告创意，根据众人的输入创造出有针对性的促销方案。另一方面，客户也能够通过自己的智能手机获得个性化的相关产品推荐，进一步激励他们反馈，强化了品牌忠诚度。

6.1.3 特殊化

从商业角度讲，地理营销的目标之一就是围绕某个位置向用户推出促销和广告。网络广告使用地理定位可以向用户提供城市级别的相关内容。而用户位置很可能是根据其设备的 IP 地址获得的，因此在城市级别上提供内容是合理的结果。毕竟，台式电脑只能在办公室或者在家里使用，无法移动。没有必要向台式电脑用户提供比城市级别更具体的广告。

对于智能手机而言，情况就完全不同了。用户在上网或访问社交媒体应用时，很可能是在移动之中。在这种情况下，更加特殊化的广告则会产生更好的效果。

将来的地理营销领域应该会出现更多位置级别的广告，这些广告都是根据地理定位为用户产生的特殊内容。随着更多 Shopkick 之类的应用面世，在用户使用应用的同时，仅基于用户所在位置的特殊推荐和内容也会越来越来普遍。对用户个体的关注，最终会带来他们对公司和品牌忠诚度。

再想一想 6.1.1 节举过的那个例子。这一次，那个人没有开车，而是步行走进一家商场。他的手机上安装了一个应用，该应用被设置为向他推荐有关运动方面的商品。当他走到距离一家足球用品专卖店大约 100 米的地方时，他收到一个提醒，告诉他这家店正在限时打折促销他最想要的拜仁慕尼黑队服球衣，印有阿扬 · 罗本的 10 号，打 75 折，现在距离促销结束还有 30 分钟。在赶往这家店的路上，他心里暗想：“怎么会这么巧呢，刚发完一条微博，说罗本的表现太棒了，帽子戏法血洗汉堡。”的确

是太巧了，不是吗？你很快就能体验到这种应用了。

6.2 地理社交

地理社交（geosocial），指的是基于地理定位的社交媒体。正是因为社交媒体使用了地理定位来促进人与人的交流，基于位置的服务才能够像现在这样大行其道。鉴于这一领域目前的繁荣景象，其未来的发展也不可小觑。地理社交应用还将继续为各类公司向用户发布广告提供新的途径，而且能够让广告更加个性化。我希望地理社交这个领域能够像社交媒体应用一样迅速发展起来，这样就可以创造众多的新机会，也能造就一批像社交媒体开发人员那样的弄潮儿。

6.2.1 持续增长

社交媒体行业已经从原来只针对美国特定用户构建的几个应用，发展成席卷全球、规模达数十亿美元的行业。社交媒体行业如果要进一步发展，将会面临一些挑战，只有克服这些挑战才能真正被普通用户接受。到了这一步，这个行业的潜在价值将是不可限量的。

应用开发重视的安全问题对应用的用户而言也同样非常重要。用户在把信息发送给应用保存之前，脑海中一定会浮现几个问题：这些数据会用什么存储方式保存、这种存储方式怎样保护数据不被外界偷窥、服务器操作系统采取了什么保护措施、服务器操作系统是否已经打上了最新的安全补丁、传输协议是否支持 TLS（Transport Layer Security，传输层安全）机制？

对于涉及用户敏感信息的任何商业行为而言，隐私数据的安全都是最最重要的。在未知对方完全可信的情况下，绝不可以授权它访问隐私数据。大多数公司都已经深刻认识到隐私的重要性，而且一直在努力采取适当方式最大限度地保护隐私信息。

作为未来社交媒体应用的代表，Ditto（hwww.ditto.me）的思维方式是超前的，具备值得社交媒体行业发扬光大的要素。从签到功能来看，Ditto 与 Foursquare 非常类似。但与传统签到应用的不同之处在于，Ditto 会向你的朋友们询问他们想做什么，而不是他们正在做什么。不信你看看 Ditto 的首页，它的主题词就是“What are you up to?”（想干什么？）

将来一定会出现模仿 Ditto 的其他应用，就像那些模仿 Yelp 或其他早期社交媒体应用的项目一样。只要社交媒体应用能够把握好不同用户的不同需求，获得更广泛的支持和使用就易如反掌了。如果真的很有用，肯定会有更多人来捧场的。最终，社交媒体将一统天下。

6.2.2 自动签到

希望未来基于位置的服务能够减少用户的手指操作。目前大家已经习以为常的签到应用是让用户搜索自己到达的商户并签到，将来的签到应用可以更加聪明、更加自动化。比如，让应用替你跟踪地理定位，根据你的位置和已有的商家数据库给出建议。用户在特定地点签到是一种众包模式，未来的应用可以根据用户在相应地点签到的地理定位来精确地判定商户的位置。使用这种应用的人越多，它的"学习"越深入，给出的建议也就越精确。

随着技术的不断进步和地理定位越来越精确，自动签到会成为应用的主流。比如，可以在手机界面上弹出一个小窗口，告知用户即将在当前位置给用户签到，请用户确认。有些应用甚至可以不必通知用户就自动完成在某一地点的签到。Shopkick（www.shopkick.com）已经在推出了自动签到功能，相信更多的应用也会紧随其后。

6.2.3 双向数据

社交媒体应用的数据一方面来自应用，另一方面来自用户。用户为应用提供信息，例如他的当前位置和关于当前位置的想法（评论和提示），等等。然后，应用据此向用户提供附近的商户和其他用户信息。很显然，社交媒体应用有赖于信息的共享。

快速发展的社交媒体公司必须面对一个最大的挑战是用户的隐私问题。市面上任何一款应用都要对相关的各种问题给出答案。怎样保护用户的位置信息？不同用户之间怎样共享彼此的信息？什么信息可以在应用中公之于众？

当然，这也需要社会公众转变对隐私的看法，尤其是在地理定位日益与我们的生活紧密相关的形势下。举个例子，一个人在公共场所的位置信息是否需要跟一个人的银行账户信息一样受到同等的保护？什么是可以公开的，什么是不能公开的，都需要公众重新思考。

如果用户提供的数据不能放量增长，社交媒体行业就难以继续发展和进步。社交媒体应用之所以对用户有帮助，关键是用户可以通过它们来改善自己的生活。要达到这个目的，唯一的办法就是应用必须有效地挖掘利用可用的用户数据。

"90 前"与"90 后"对隐私的看法就不太一样，后者对社交媒体在生活中的作用有更深入的理解，或许是无意识的，因为发短信和发微博已经成了他们生活中不可分割的一个重要部分了。这一代人对隐私的看法已经发生了变化。像我们这些人就得尽力追赶他们的脚步，也像他们那样学会解放思想，重新认识什么是隐私。只有这样才能开发出更有价值的应用。社交地理应用与签到应用相比，可以让你的朋友更详细地知晓你身在何方，有望成为每个人社交经历中不可或缺的一部分。

数据是一条双向的马路，只要两个方向都畅通，社交媒体应用才能大行其道。应用可以接收到的数据越多才能做得越好，对人们日常生活的作用才会越大。若要接收数据，必先给出数据。

6.3 地理标签

地理标签（geotagging）指的是向图像、视频、文本消息、微博及网页等数字媒体添加地理空间元数据。在这些元数据中，最常捕获到的地理空间信息就是经度、纬度、精确度和前进方向。媒体不同，取得的元数据也会有一些差别。看过后面的介绍你会发现，这些地理空间信息与通过 W3C Geolocation API 取得的数据基本相同。

6.3.1 数字媒体与地理标签

Twitter 在 2009 年夏天开始支持地理标签，但完整的 API 直到 2009 年 11 月才推出。这个功能在用户偏好设置中默认是禁用的，这样就对使用地理标签的 Twitter 用户提供了事前同意的机会。要想启用位置功能，可以在用户设置中打开“Account”（账号）选项卡，然后勾选“Tweet Location”（推文位置）复选框。Twitter 指出，用户一旦启用了地理标签功能，它就会保存相应的位置信息，而用户随时都可以删除自己的历史位置。

YouTube 是 2007 年夏天开始支持地理标签的。向 YouTube 上传新视频时，可以在“Video Upload”（上传视频）的“Broadcast Options”（传播和分享选项）右侧看到一个“Date and Map Options”（数据与地图）区。在这里，可以指定纬度和经度坐标或者在提供给你的 Google 地图上添加一个点。如果是指定坐标，应该按照下面的格式：

```
geo:lat=38.624722 geo:lon=-90.185278
```

很多图片共享网站也支持数字图像的地理标签。图片共享网站有很多，比如 Flickr、Photobucket、Picasa、Photoworks、Twitpic、Snapfish、Shutterfly、Fotki，等等。主要的图片共享网站已经认识到支持地理标签的重要性，因为越来越多的照片都是由支持地理标签的手机内置的相机拍摄的。随着更多相机内置支持 GPS，地理标签有望成为数字图片的标准特性。

6.3.2 隐私与地理标签

在发微博、发视频或发图片的时候共享地理信息，自然会让人想到将自己的位置在网上公之于众会不会涉及隐私。发微博的时候，你是在哪里发的那 140 个字符与拍

完自己家里的照片被人知道你在哪里不能同日而语。视频和图片中所包含信息的隐秘性确实是值得关注的。因此，在面向社会公众的站点上，采取适当的安全措施，将某些媒体内容标注为私人信息是极其重要的。

不过，用户有时候也要确认一点，即自己是否信得过某些内容提供商能够恰当处理和保护他们的内容。如果答案是否，那用户应该自己主动禁用在自己的微博或视频中添加地理标签，而且要从上传到公众站点的图片中删除地理信息。总之，媒体网站只提供服务，至于用户想使用和添加什么服务，都应该由用户自己说了算。

6.4 地理应用

地理应用（geo-application）就是有某项功能依赖于地理定位的应用。前面讨论的社交媒体应用就是一种地理应用。当然，还有其他很多应用可以利用地理定位。比如 CitySourced（www.citysourced.com），它致力于培养公民的责任感和参与精神，让人们能够把哪里路面上有坑坑洼洼、哪里有人乱写乱画、哪里有人乱扔垃圾等信息提交到网上。随着我们不断把技术进步与日常生活结合起来，诸如此类利用地理定位的应用还会越来越多。

6.4.1 安全/跟踪

将来有可能出现地理定位应用的一个领域是安全，特别是少年儿童的安全。只要你的孩子随身携带了一台能够发射信号的 GPS 设备，这种应用就可以随时知道他在哪里。利用这种应用，父母可以通过监视孩子的 GPS 位置来保证他们去了应该去的地方。不妨大胆地预测一下，到时候孩子携带在身上的 GPS 设备可能会极小（可以缝在他们的衣服里，甚至注入到他们的皮肤下面）。

类似的跟踪技术还有别的用途。徒步旅行者以及滑雪爱好者可以穿上内置了 GPS 发射器的衣服。这样，万一迷路或者遭遇雪崩等自然灾害就可以为营救他们提供便利。跟踪军队作战也是一项现实的应用。皮下 GPS 装置可以在战士深入敌后被俘或失踪的时候，辅助部队找到他们。总之，无论是民用还是军用，这种设备都能为营救工作提供便利。

6.4.2 打车

地理定位技术的另一项应用也很有前景，那就是确保在城市郊区能打到出租车，或者说随处可以打车。如果你可以把自己的位置发送到出租车服务，那该服务就可以告诉你距离最近的交通工具（因为该服务通过 GPS 跟踪这些交通工具），从而节省你等待的时间。当然，不一定只是为了打车。旧金山的一家创业公司 Uber（www.

uber.com）通过 Android 或 iPhone 应用与豪华轿车司机建立联系，根据用户的位置能够找到距离最近的司机。将来这种服务一定会更加大众化。

6.4.3 搜索

如果有地理定位，那么搜索服务就能够更加个性化。即便是在台式计算机而非智能手机上，同样也能够实现个性化。所谓个性化，就是用户在搜索某个关键词的时候，搜索引擎可以在必要时考虑用户的地理定位，并返回与位置有关的一些结果。这样与使用以往的权重评估机制相比，能够更快地为用户给出搜索结果。举个例子，打开最新的搜索引擎 GeoFind，输入 family owned restaurant。这个搜索引擎会在前面列出你所在地区所有的家庭餐馆。而在一般的搜索引擎里，首先映入眼帘的则是维基百科上的 family owned restaurant 条目，然后是一些你根本不关心的结果，再往后才是你想找的家庭餐馆信息。而且，即便搜到了家庭餐馆，也大多是其他州的餐馆。如果在搜索过程中考虑地理定位的因素，那么结果就完全不同了。

6.4.4 移动商务

移动商务（M-Commerce）就是在智能手机等移动设备上实现商业交易。Shopkick 等应用已经在探索如何将地理定位与移动商务结合起来了。假以时日，必将有更多利用地理定位的移动应用问世，让用户可以根据自己的所在位置进行购物。

6.4.5 其他应用

其他形式的应用也可以在实现其功能的同时利用地理定位。可以在旅行应用中利用它实现导航，可以在社交媒体中利用它实现位置规划。还可以在健身应用中用到它，比如跟踪你锻炼身体的过程。只要能想到地理定位的某种新用途，就有可能产生一种新的应用。未来基于地理定位的应用将会越来越多。

6.5 HTML5与地理定位

HTML5 和 W3C Geolocation API 的未来前景不可限量。本机地理定位应用正日益得到消费者的认同，而且没有证据表明这个趋势有什么变化。这对于基于 Web 的地理定位应用同样是一个好消息。HTML5 也日益深入人心，随着主要浏览器开发商对它的支持越来越完善，开发人员的道路也将越来越宽广。把 HTML5 与利用 W3C Geolocation API 的浏览器应用结合起来，就可以创建出堪与本机应用媲美的浏览器应用。

6.5.1 辅助LBS的Web应用

可以利用 HTML5 实现对移动地理定位应用的扩展。我们拿 Twitter 举例，通过 Twitter 的源可以实现查看到某个坐标附近区域最近发表的所有推文：在地图上点击某个坐标点，然后所有在该位置 500 米范围内的推文就会显示出来。更令人兴奋的是，你可以利用 W3C Geolocation API 取得你所在位置附近 500 米内的所有推文，如图 6-1 所示。程序示例 6-1 展示了这个例子的代码。

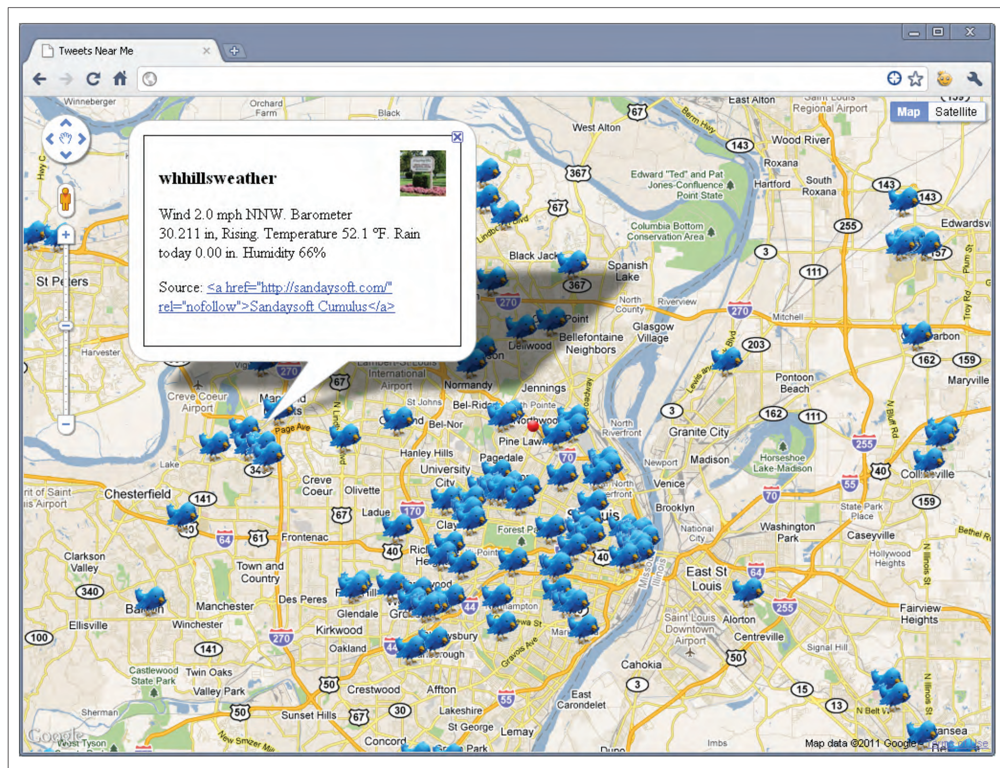


图 6-1　综合使用 W3C Geolocation API 和 Twitter Search API

程序示例 6-1　利用地理定位使用 Twitter Search API 的应用

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8"/>
    <meta http-equiv="X-UA-Compatible" content="IE=7"/>
    <meta http-equiv="viewport" content="initial-scale=1, maximum-scale=1,
       user-scalable=no"/>

    <title>Tweets Near Me</title>
```

```
  <style type="text/css">
  html { height: 100% }
  body { height: 100%; margin: 0; padding: 0 }
  #map { height: 100% }
  .tweet_info { border: 1px solid #000; padding: 15px; width: 300px }
  .tweet_info img { float: right; height: 48px; margin: 0 0 10px 10px;
      width: 48px }
  .tweet_info h3 { margin-bottom: 10px }
</style>

<script type="text/javascript"
    src="http://maps.google.com/maps/api/js?sensor=true"></script>
<script type="text/javascript"
    src="https://ajax.googleapis.com/ajax/libs/jquery/1.5.2/jquery.min.js">
</script>
<script type="text/javascript" src="jquery.timer.js"></script>
<script type="text/javascript">
  var map = null;
  var browserSupport = false;
  var attempts = 0;
  var tweets = [];
  var tweetsQ = [];
  var refreshQuery = '?q=';
  var infoWindow = new google.maps.InfoWindow();

  /* 这个函数在页面加载完毕后立即执行 */
  function initMap() {
    /* 设置地图的所有选项 */
    var options = {
      zoom: 4,
      center: new google.maps.LatLng(38.6201, -90.2003),
      mapTypeId: google.maps.MapTypeId.ROADMAP
    };

    /* 创建一幅新地图 */
    map = new google.maps.Map(document.getElementById('map'), options);

   /* 设置计时器以收集推文 */
  $(document).everyTime('30s', acquireTweets);
  $(document).everyTime('100ms', parseTweetsQ);

  /* 添加地理定位功能 */
  getLocation();
}
/*
 * 如果浏览器支持 W3C Geolocation API
 * 则取得当前位置；否则，报告问题
 */
function getLocation() {
  /* 检查浏览器是否支持 W3C Geolocation API*/
  if (navigator.geolocation) {
    browserSupport = true;
    navigator.geolocation.getCurrentPosition(function(position) {
        plotLocation(new google.maps.LatLng(position.coords.latitude,
            position.coords.longitude));
```

```
    }, reportProblem, { timeout: 45000 });
  } else
    reportProblem();
  }

  /* 创建用于调用 Twitter Search API 的 URL*/
  function createTweetSearchURL() {
    var temp = map.getCenter();

    return 'http://search.twitter.com/search.json' + refreshQuery +
        '&geocode=' + temp.lat() + '%2C' + temp.lng() +
        '%2C50km&rpp=100&callback=?';
  }

  /* 把地图上标注位置并适当放大，然后取得推文 */
  function plotLocation(latLng) {
    attempts = 0;

    map.setCenter(latLng);
    map.setZoom(11);

    var marker = new google.maps.Marker({
      position: latLng,
      icon: 'http://geo.holdener.com/images/myloc.png',
      animation: google.maps.Animation.DROP
  });
  marker.setMap(map);

  acquireTweets();
}

/*
 * 调用 Twiter Search API 并遍历结果
 * 将推文依次放到推文队列中
 */
function acquireTweets() {
  $.getJSON(createTweetSearchURL(), function(data) {
    if (data.results)
      $.each(data.results, function(i, tweet) {
        if (tweet.geo || tweet.location)
          tweetsQ.push(tweet);
      });
    refreshQuery = data.refresh_url;
  });
}

/* 解析推文队列并标注有坐标的推文 */
function parseTweetsQ() {
  if (tweetsQ.length > 0) {
    var tweet = tweetsQ.pop();

    /* 检查是否有坐标信息 */
    if (tweet.geo) {
      tweet.latlng = new google.maps.LatLng(tweet.geo.coordinates[0],
          tweet.geo.coordinates[1]);
```

```
      plotTweet(tweet);
    }
  }
}

/* 创建在弹出窗口中显示的内容 */
function createInfoContent(tweet) {
  var retval = '';

  retval += '<div class="tweet_info">';
  retval += '<img alt="' + tweet.from_user_id + '" src="' +
      tweet.profile_image_url + '" class="tweet_profile"/>';
  retval += '<h3>' + tweet.from_user + '</h3>';
  retval += '<p>' + tweet.text + '</p>';
  retval += '<p>Source: <a href="' + tweet.source + '"/>' +
      tweet.source + '</a></p>';
  retval += '</div>';
  return retval;
}

/*
 * 在地图上插入推文，添加 click 事件
 * 以便显示 infoWindow
 */
function plotTweet(tweet) {
  tweet.marker = new google.maps.Marker({
    position: tweet.latlng,
    icon: 'http://geo.holdener.com/images/tweet.png',
    animation: google.maps.Animation.DROP,
    title: tweet.from_user,
    html: createInfoContent(tweet)
  });
  google.maps.event.addListener(tweet.marker, 'click', function() {
    infoWindow.setContent(this.html);
    infoWindow.open(map, this);
  });
  tweet.marker.setMap(map);
  /*
   * 如果地图上的推文超过 100 条
   * 则删除其中最老的推文
   */
  if (tweets.length > 100) {
    var tweet = tweets.shift();

    tweet.marker.setMap(null);
  }
}

/* 使用这个函数来报告错误 */
function reportProblem(e) {
  /* 判断是浏览器支持问题，还是 API 自身的问题 */
  if (browserSupport) {
    switch (e.code) {
      case e.PERMISSION_DENIED:
        alert('You have denied access to your position. You will'+
```

```
            'not get the most out of the application now.');
          break;
        case e.POSITION_UNAVAILABLE:
          alert('There was a problem getting your position.');
          break;
        case e.TIMEOUT:
          /* 在最终判定超时之前，再给三次机会 */
          if (++attempts < 3) {
             navigator.geolocation.getCurrentPosition(plotLocation,
                reportProblem);
          } else
            alert('The application has timed out attempting to get ' +
                'your location.');
          break;
        default:
          alert('There was a horrible Geolocation error that has ' +
            'not been defined.');
        }
      } else
        alert('Geolocation is not supported by your browser.');
    }

    $(document).ready(initMap);
  </script>
</head>
<body>
  <div id="map"></div>
</body>
</html>
```

以上代码开始的时候类似第 4 章 Google 地图的例子，只是多了三条样式规则，用于给应用中的信息窗口设置样式。在加载 Google API 之后，从 Google CDN（Content Delivery Network，内容分发网络）加载最新的 jQuery 库。然后再从本地加载 jQuery Timers 插件。

页面加载完毕后，应用会基于大多数默认选项来创建一幅 Google 地图。接着，设置每 30 秒调用一次 `acquireTweets()` 函数，每 100 毫秒调用一次 `parseTweetsQ()` 函数。其中对 W3C Geolocation API 的使用与其他例子中相同。

通过 W3C Geolocation API 成功取得位置信息后，将地图居中于该位置，并使用自定义图像及 `google.maps.Animation.DROP` 动画在对应坐标点创建一个标记。然后，以手工方式初次调用 `acquireTweets()` 函数。

`acquireTweets()` 函数使用 jQuery 的 `$.getJSON()` 方法调用 Twitter Search API。在这个函数中，通过 `geocode` 参数向 Twitter Search API 传入了通过 W3C Geolocation API 取得的纬度和经度，将范围设置为 50 公里。每次调用返回 100 个结果——通过 `rpp`（result per page，每页结果数）参数指定。得到的推文都放到名为 `tweetsQ` 的

数组中，这个数组将作为待处理推文的队列。调用 Twitter Search API 时可以传递的所有选项可以查阅其在线文档。

Twitter Search API 会返回一个 JSON 对象。这个对象的结构类似如下所示：


```
{
   "results": [
     {
        "from_user_id_str" : "111111111",
        "location": "St. Louis, MO",
        "profile_image_url":
            "http://a1.twimg.com/profile_images/111111111/my_profile_pic.gif",
        "created_at": "Wed, 20 Apr 2011 16:11:05 +0000",
        "from_user": "geotagged01",
        "id_str": "99999999999999901",
        "metadata": {
          "result_type": "recent"
        },
        "to_user_id": 222222222,
        "text": "@geotagged02 Tweet for HTML5 #Geolocation",
        "id": 99999999999999901,
        "from_user_id": 111111111,
        "to_user": "geotagged02",
        "geo": null,
        "iso_language_code": "en",
        "to_user_id_str": "222222222",
        "source": "&lt;a href=&quot;http://twitter.com/&quot;&gt;web&lt;/
           a&gt;"
      },
      .
      .
      .
      {
        "from_user_id_str": "222222222",
        "location": "STL",
        "profile_image_url":
            "http://a1.twimg.com/profile_images/222222222/my_profile_pic.jpg",
        "created_at": "Wed, 20 Apr 2011 16:11:05 +0000",
        "from_user": "geotagged02",
        "id_str": "99999999999999902",
        "metadata":{
          "result_type": "recent"
        },
        "to_user_id": 333333333,
        "text": "@geotagged03 Another HTML5 #Geolocation tweet",
        "id": 99999999999999902,
        "from_user_id": 222222222,
        "to_user": "geotagged03",
        "geo": null,
        "iso_language_code": "en",
        "to_user_id_str": "333333333",
        "source": "&lt;a href=&quot;http://twitter.com/&quot;&gt;Twitter \
            for Android&lt;/a&gt;"
```

```
    }
  ],
  "max_id": 99999999999999909,
  "since_id": 99999999999999902,
  "refresh_url": "?since_id=99999999999999909&q=",
  "next_page": "?page=2&max_id=99999999999999909&rpp=100&geocode=
     38.111111%2C \ -90.555555%2C50.0km&q=",
  "results_per_page": 100,
  "page": 1,
  "completed_in": 0.162742,
  "warning": "adjusted since_id to 99999999999999902 (), requested since_id \
      was older than allowed -- since_id removed for pagination.",
  "since_id_str": "99999999999999902",
  "max_id_str": "99999999999999909",
  "query": ""
}
```

返回结果后，acquireTweets() 函数会循环遍历 data.results 数据，把每条带有地理标签或者已经由用户设置了位置信息的推文放到 tweetsQ 队列中。

随后，处理这些推文的 parseTweetsQ() 函数每 100 毫秒就会执行一次，每次从队列中拿出一条推文。它还要检查每条推文是否带有地理编码，如果有则将推文发送并绘制到地图上。相应地，plotTweet() 函数基于用户发送每条推文的坐标在地图上放置一个自定义图像作为标记。单击标记就会显示与相应推文相关的信息。如果地图上的推文超过了 100 条，则删除最老的推文。

而 reportProblem() 函数与 Google 地图例子中的同名函数一样，会在 Geolocation API 没有取得位置的情况下报告问题。阅读 Twitter API Docucment 可以了解如何在混搭应用开发中更好地利用 Twitter 的功能，而不只是简单地向用户罗列数据。

同样，利用 Foursquare 也可以通过 Web 应用实现辅助移动应用。Where Do You Go（www.wheredoyougo.net）能够显示你去过的地方的热区图，4mapper（4mapper.appspot.com）与 weeplcaes（www.weeplaces.com）也实现了类似功能，但后者还添加了非常有用的时间线，在地图上可视化地显示签到信息。Fourwhere（fourwhere.com）则集成了对 Foursquare、Yelp 和 Gowalla 的搜索，能够基于在地图上点击的坐标点显示一定范围内的位置信息。

对于希望看到大量信息的用户来说，将数据可视化是一种很有用的方式，而把数据直观地显示在地图上效果更好。HTML5 也支持以图表形式显示信息，那是另一种形式的数据可视化。总而言之，基于 HTML5 的 Web 应用是对相应移动设备本机应用的非常好的补充。希望将来可以看到更多这种类型的应用问世。

6.5.2　基于Web的LBS

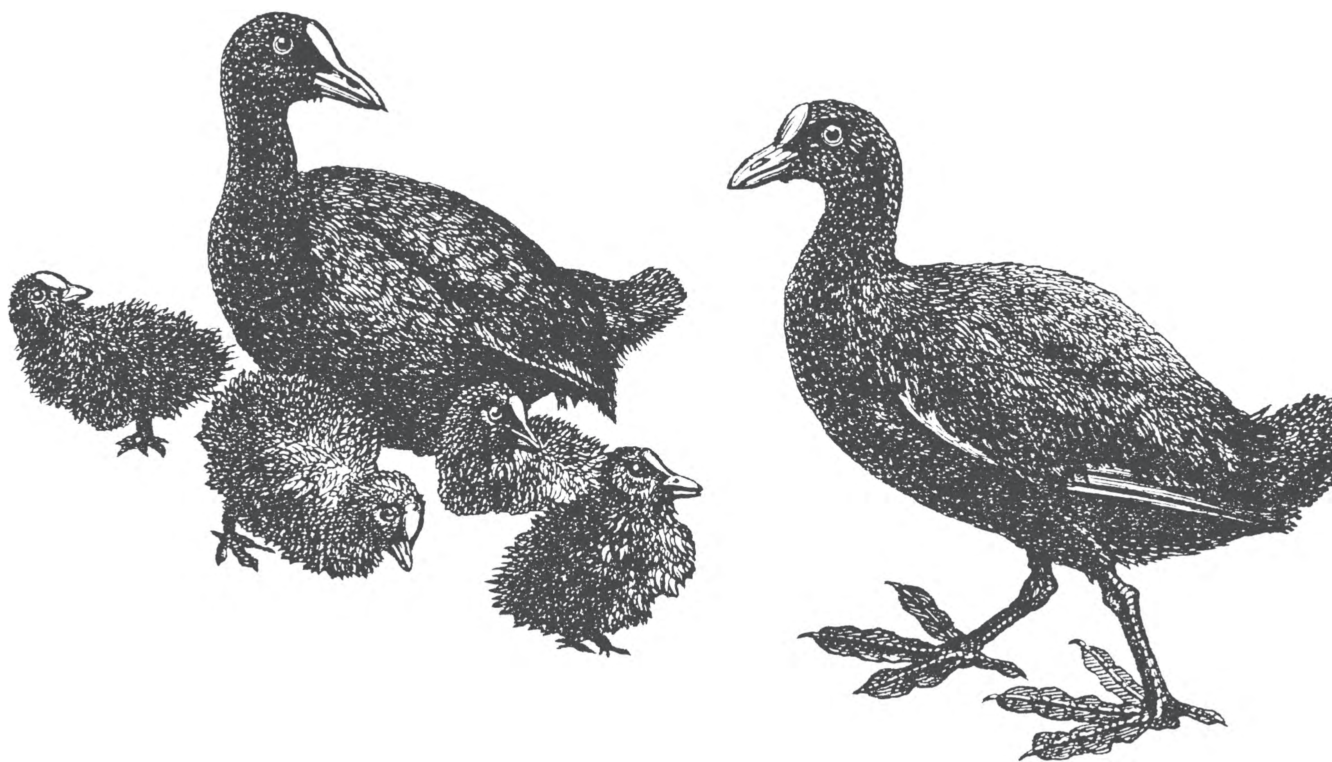
既然现在有了 W3C Geolocation API，那么本书所讨论的所有移动应用中，大部分都没有理由不能以 Web 应用的形式出现。增强现实应用还不能在浏览器中通过 JavaScript 来编写，但像 Foursquare 这样的社交媒体应用写成 Web 应用的形式是完全有可能的。虽然把已有的这些应用重新用 JavaScript 写一遍，推出相应的 Web 应用版没有多大意义，但新的地理定位应用多数情况下都是可以考虑直接针对浏览器来开发的。

针对浏览器开发有不少优势。首先，整体开发成本比较低，因为只要使用 HTML、CSS、JavaScript 和某种服务器端脚本语言编写一次代码就行了，而不必针对每种移动平台都开发一套不同的代码。另外，客户端也无需特别配置，Web 应用就是在浏览器中运行的。任何更新只要在服务器上做一次，就能在用户下次通过浏览器加载页面时自动升级。也就是说，Web 应用永远是最新版本。

在不久的将来，浏览器还有可能成为你所用设备的操作系统，你对设备的使用都要通过浏览器。到了那个时候，针对浏览器开发地理定位应用的经验能够让你处于优势地位。当然，W3C Geolocatioin API 还比较新，学习掌握它还需要时间，但针对浏览器开发的基于位置的应用一定会与日俱增。希望有朝一日，利用 HTML5 和 W3C Geolocation API 的革命性的地理定位应用能够横空出世。地理定位时代的帷幕已经拉开，好戏就要开场了！

HTML5：等轴实时游戏开发

［阿根廷］Mario Andrés Pagella 著



前言

上瘾，沮丧；好玩，无聊；着迷，重复；轻松，费劲。

上面每对词的意思看起来都是矛盾的，但作为一名即时战略游戏的玩家，我认为它们可以表达出我那如同坐过山车一般的心情。当初，在 EA/Maxis 的《模拟城市》（*SimCity*）、《模拟城市 2000》，Chris Sawyer 的《运输大亨》（*Transport Tycoon*），还有 Bullfrog Productions 的《主题医院》（*Theme Hospital*）等这些令人心醉的游戏上面，我花了数不清的时间。我一直很纳闷，为什么我的朋友当中只有几个人（都是那种极客型的人）玩过这些游戏。

然而现下的情况是，无论从小孩子到初中生，从老太太到足球妈妈，从守旧的兄弟会成员到计算机极客，几乎都在 Zynga 的《开心农场》（*FarmVille*）或《城市小镇》（*CityVille*），Playdom 的《社交城市》（*Social City*）或 Playfish 的《我的帝国》（*MyEmpire*）这些游戏上花过几个小时。但他们并不知道这些游戏还有祖先，那要追溯到等轴实时游戏[1]的黄金时代。但恐怕他们决不会去碰当时的那些游戏。

这个世界怎么了？

等轴实时游戏最近得以爆发，一方面是因为 Zynga 在他们推出的这种游戏中做到了“去粗取精、去伪存真”，另一方面则是由于消费者的兴趣发生了变化。这些游戏不会再让你因为没人“搬到你的城市”（《模拟城市》）而苦恼，相反，你可以把朋友加为邻居。这些游戏利用了 Facebook 的社交功能改变了游戏的性质。结果，原先的无聊之处现在变得更具有交互性了，你不仅可以摆放物体，还能亲手创建它们，而且

注 1：原文为 Isometric social real-time Game，关于 Isometric Game 一词的翻译请参看译者的文章“Isometric Game 及译法漫谈”（www.ituring.com.cn/article/788）。（译者注）

还可以手工收集由此而得到的分数。经过一段时间（一般几周）之后，当你觉得游戏没什么可玩的时候，游戏还会给你一些任务，让你跟自己的朋友绑定在一起。最终，即便你不再玩这个游戏，你构筑的建筑物也会被保留，并将继续创造效益和分数。（在游戏业界，这种思想被称为*异步交互*或*异步游戏机制*。）

一旦摆脱了挫折、乏味和重复（等轴实时游戏的三大杀手），那么这种游戏就会摇身一变，让人感觉上瘾、好玩、着迷、轻松（也可能是费劲，这要看你想怎么玩了）。加之跟社交有关的进度要求，转瞬之间它们就可以像病毒一样扩散得无所不在。说到这里，要理解 Zynga 在本书写作时的估值高达 100 亿美元（超过了世界上最大的“老牌”游戏公司 EA），或者迪士尼花 7.6 亿美元购买 Playdom 就不足为奇了。很难对影响游戏可玩性的各个方面的重要性作出准确的估计。但是，只要尽力把各方面都做到尽善尽美，那么这种社交游戏就有可能一炮走红。

与以前的即时战略游戏相比，等轴实时社交游戏的界面比较简单：就是一个“活动”的地图编辑器。你可以在区块矩阵（通常把它叫做“网格”）上摆放任何物体。而有些物体——在我们这里是建筑物——可以在每 T 时间内生成 P 点分数。换句话说，即便我们不玩游戏，建筑物仍然能生成分数。

我们最终将完成一个示例项目，在这个项目中，我们会开发一款名为《旅游胜地》（Tourist Resort）的游戏。用户在游戏中必须在某个地方建设一个度假村，以各种树作为装饰，还要摆放不同的店铺。每个店铺每 T 时间要生成 N 个金币；而用户拿这些收益又可以去购买更多的建筑。

HTML5来了

在等轴社交游戏不断进步的同时，开发这些游戏的技术也在发生变化。

多年以来，开发在 Web 浏览器中运行的精美且具有高度交互性的游戏会用到 VRML（Virtual Reality Modeling Language，虚拟现实建模语言）、Java Applet、Macromedia Shockwave、Adobe Flash、Microsoft Silverlight、Unity3D 等，这些全都是第三方、专有的解决方案，而且为了运行用它们开发出的游戏，用户还必须下载安装浏览器插件。更有甚者，为了开发这些游戏，开发人员不得不花大价钱购买 IDE。

然而，运用 HTML、CSS 和 JavaScript 等 Web 技术开发的游戏体验，又难以跟 Adobe Flash 等工具开发的体验相媲美。加上浏览器——特别是 JavaScript 的运行速度慢、不原生支持视频、音频和本地存储，而且其中有的浏览器，比如 Internet Explorer 既不支持透明 PNG 图像，又没有给开发者提供工具去实现哪怕是最基本的位块传输（bit-block transfer）功能。由于以上种种情况，Web 技术除了开发最简单

的游戏，确实乏善可陈。

幸运的是，随着时间的推移，大多数主流浏览器都逐步实现了 HTML 和 CSS 的最新版本：HTML5 和 CSS3。与此同时，更大大提升了 JavaScript 引擎的性能。今天，Mozilla Firefox、苹果 Safari、谷歌 Chrome、Opera 以及微软 Internet Explorer 9 等浏览器的最新版本，包括 iPhone、黑莓手机还有基于 WebOS、Android 的手机等智能终端中的浏览器，都已经实现了开发全功能视频游戏必需的大部分技术。

基本要求

我们不会面面俱到地介绍 HTML5、CSS3 或 JavaScript，因此需要读者至少掌握这些语言的基本知识。在此基础上，这部分各章将会围绕高效地利用这些技术，讲解针对所有智能手机、平板电脑或者电脑中支持 HTML5 的浏览器开发游戏。

这部分适合想要尝试开发游戏的 Web 开发人员，也适合想把自己的经验移植到 Web 平台的游戏开发人员。

我们将要利用等轴投影原理开发一款融入社交元素的即时战略游戏，这款游戏的目标平台是上述所有平台的"最小公分母"——移动设备。显然，如果这款游戏能够在移动设备中以适当的速度运行，那它就一定可以在 PC 等更强劲的设备上运行。

代码示例

所有代码示例及示例的支持文件都可以在如下地址下载到：https://github.com/andrespagella/Making-Isometric-Real-time-Games。

开发及调试工具

有的读者或许已经是经验丰富的开发人员了，但即便如此也需要一些重要工具。示例使用简单的文本编辑器（记事本或 TextEdit）就可以编写，在支持 HTML5 的浏览器中就可以运行。不过，如果你想正式地做一些开发，那最好还是使用语法高亮、JavaScript 控制台、JavaScript 调试器和 Web 检查器等工具。为此，我建议大家选用支持（或通过扩展方式支持）JavaScript、HTML 和 CSS 的编辑器，比如 vim 或 emacs。

JavaScript 控制台、JavaScript 调试器和 Web 检查器是用来查找和追踪问题、例程或对象的工具。好在，大多数主流浏览器都为我们内置了这三个助手。

Mozilla Firefox

在"工具"菜单中，可以找到"JavaScript 控制台和检查器"。但我还是要建议大家

安装 Firebug，也就是 Firefox 的扩展。这个扩展特别适合进行高级的 Web 开发，也包含了 JavaScript 调试器和 HTML、CSS 及 DOM 检查器，当然还有其他很多功能。

Internet Explorer

按 F12 键打开“开发人员工具”窗口，里面包含一个 JavaScript 控制台，可以在不同的“文档模式”下查看页面，让你知道自己的站点在 IE5、IE7、IE8 中显示的效果。

谷歌 Chrome

打开“查看”菜单（在 OS X 版本中），或者单击右上角地址栏右侧的小扳手图标，就可以看到“工具”子菜单，点击该菜单可以看到“开发人员工具”和“JavaScript 控制台”。这两个工具是基于 WebKit 的浏览器共有的。

Safari

在“偏好设置”的“高级”选项卡中，勾选“在菜单栏中显示‘开发’菜单”，即可访问各种开发工具。在 OS X 中，要显示“开发”菜单，需要在终端窗口中执行命令：`defaults write com.apple.Safari IncludeDebugMenu 1`。另一种方法是编辑 Preferences.plist 文件，在 XML 结束标签 `</dict>` 和 `</plist>` 前面添加下面这一行：`<key>IncludeDebugMenu</key><true/>`。视版本不同，OS X 中的 Preferences.plist 文件可能在如下位置：

- *C:\Documents and Settings\\`%USER%`\Application Data\Apple Computer\Safari* 中，其中 `%USER%` 是你的账户名；
- *C:\Users\\`%USER%`\AppData\Roaming\Apple Computer\Safari* 中，其中 `%USER%` 是你的账户名。

在 Windows 系统中，可以编辑启动的 Safari 快捷方式，在路径中 Safari.exe 的后面添加 `/enableDebugMenu` 参数。

Opera 10

Opera 也提供了一套非常棒的调试工具和 Web 检查器，这套工具就是 Dragonfly。要了解详细信息，请参考 Opera Dragonfly 的网站：http://www.opera.com/dragonfly/。

关于游戏设计

游戏设计是游戏开发最重要的环节之一——没有人想玩一款无聊的游戏。在融入社交元素的即时战略游戏中，要想让用户玩得开心、投入，除了让游戏的用户体验好、好玩之外，还得尽可能多地让游戏与用户的社交网络及经验集成起来。

不要忘了这类游戏的主要诉求（这当然是显而易见的），那就是让用户与自己的朋友比赛（“嘿，我的城市比你的大 / 比你的好！”）。来自朋友们的推荐是最好、最有说服力的广告。

当然，还需要认真负责地对待用户的社交关系。被社交网络（比如 Facebook）禁止无疑是灾难性的，然而在此之前，你的游戏更有可能被玩家屏蔽掉，或者被贴上“垃圾”的标签！

游戏设计是一个十分宽泛的主题，这里不可能深入探讨。为此，我向读者推荐一本涵盖 Web 应用与游戏设计的好书，即 Gabe Zichermann 与 Christopher Cunningham 合著的 *Gamification by Design*（O’Reilly）。

致谢

谢谢我的 fiancée Regina，我的父亲 Rubén，我的其他家人，还有我最要好的朋友以及我的同事。还要感谢 Minor Studios 的所有人，尤其是 CEO Martín Repetto 跟 Xavier Amado。当然，还得感谢给了我极大帮助的编辑 Simon St.Laurent，感谢 Shelley Powers 提出的每一条技术审读意见，还要感谢对出版做出贡献的所有 O’Reilly 人。

第 1 章

图形基础：画布与精灵



HTML5 的 canvas 元素为创建复杂的图形游戏铺平了道路，不仅提供更大的灵活性，而且与通过 DOM（Document Object Model，文档对象模型）移动图像的旧方法相比，速度也快了很多。有了这个 canvas 元素，就可以直接用 JavaScript 在屏幕上绘制图形，把以往在开发游戏中使用图形的方法运用到 Web 环境中。尽管 canvas 是最近才加入到 HTML 大家庭的元素，但它已经得到了较新的桌面和移动浏览器的广泛支持。

1.1 使用canvas对象

在 canvas（画布）元素上可以通过 JavaScript 极快地绘制屏幕区域，而且能够达到像素级的操作精度。然而，canvas 工作在即时模式下，这一点与 SVG（Scalable Vector Graphics，可伸缩矢量图形；本书未作介绍）不一样，对 HTML5 Canvas API 的调用都会直接在画布上体现出来，绘制并显示出来之后不会保存对其内容的任何引用。比如，假设我们要把图形向右移 10 像素，必须先清除画布上显示的所有内容，然后再基于新坐标把它们重新绘制到画布上。稍后我们会讨论一种名为“区块适配更新”（Adaptive Tile Refresh）的技术，该技术可以避免清除整块画布，而只更新其中有变化的部分。

可以把 canvas 对象想象成一张纸，而我们有很多蜡笔（和其他绘画工具）可以在上面绘制景物。如果想画一条红线，就拿起红色蜡笔画这条线。如果想画一条绿线，就拿起绿色蜡笔画。同样，对于绘制“样式”也是如此。如果你想绘制一条从左上角到右下角的 45°线，可以直接绘制，也可以把纸顺时针旋转 45°，再从上到下画一条竖直的线（显然，直接绘制的效率更高）。

使用 HTML5 Canvas API 非常简单。首先，要在页面中添加新的 HTML5 canvas 标签，并为它指定 id 属性：

```
<canvas id="game" width="100" height="100">
    Your browser doesn't include support for the canvas tag.
</canvas>
```

位于 canvas 标签中的文本会在不支持它的浏览器中显示。稍后，我们会学习使用 JavaScript 库 Modernizr 来更有效地处理类似的不兼容性问题。



你需要在 canvas 标签中指定 width 和 height 属性。即便可以通过 CSS 将画布的宽度和高度设置为某些值，但当用 JavaScript 引用画布对象时，它还会恢复到默认大小（300×150 像素），也就是会完全覆盖通过 CSS 指定的大小。不过，倒是可以在 JavaScript 中动态地修改画布对象的宽度和高度。

为了使用 HTML5 Canvas API，第一步要通过其 id 属性的值（myCanvas）引用 canvas 元素，通过这个引用可以取得对 2D 绘图上下文的引用（WebGL 是“3D 上下文”，本书未作介绍）。

```
window.onload = function () {
  var canvas = document.getElementById('game');
  var c = canvas.getContext('2d');
}
```

另外，也可以像下面这样动态创建 HTML5 画布对象：

```
window.onload = function () {
  var canvas = document.createElement('canvas');
  var c = canvas.getContext('2d');
}
```

在前面的示例代码中，2D 绘图上下文的引用保存在了变量 c 中（其他示例中，这个变量的名字是 ctx）。对 Canvas API 的所有后续调用，都要通过这个变量来完成。本书第一个例子是用户打开游戏后最先看到的界面——*标题界面*（Title Screen）。稍后，我们会扩展这个例子，在显示标题界面的同时预先加载资源（图像、声音等）。

标题界面将占据整个浏览器窗口，其中包括一幅图像，也就是游戏的标志。标志下方是一句话：“单击或轻触屏幕开始游戏。”点击了浏览器的窗口后，标题界面会慢慢地淡出成白色。

开始之前，需要先准备一些游戏中会用到的基本 HTML 代码。基本上，这种页面就是一个普通的 HTML5 页面：

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>Example 1 - Title Screen</title>

    <script>
      //JavaScript 代码放在这里
    </script>
    <style type="text/css" media="screen">
      html { height: 100%; overflow: hidden }
    body {
        margin: 0px;
        padding: 0px;
        height: 100%;
      }
    </style>

  </head>
  <body>
```

```
    <canvas id="game" width="100" height="100">
      Your browser doesn't include support for the canvas tag.
    </canvas>
  </body>
</html>
```

代码中的 CSS 样式用于把页面设置成跟屏幕百分之百一样大，其中的 `overflow:hidden` 用于阻止浏览器在页面内容超出屏幕可见区域时显示垂直或水平滚动条。

好，模板已经有了。接下来，可以把使用 HTML5 Canvas API 的 JavaScript 代码添加到 `<script>` 标签中了：

```
window.onload = function () {
  var canvas = document.getElementById('game');

  // 强制画布随着浏览器窗口大小的变化动态地
  // 改变自身大小，与窗口保持相同宽度和高度
  canvas.width = document.body.clientWidth;
  canvas.height = document.body.clientHeight;

  var c = canvas.getContext('2d');

  // 给屏幕填充黑色背景
  c.fillStyle = '#000000';
  c.fillRect (0, 0, canvas.width, canvas.height);

  var phrase = "Click or tap the screen to start the game";
  c.font = 'bold 16px Arial, sans-serif';
  c.fillStyle = '#FFFFFF';
  c.fillText (phrase, 10, 30);
}
```

接下来我们逐步讲解一下以上代码。

在把 `canvas` 元素添加到 HTML 代码中时，我们把它的 `height` 和 `width` 属性都设置成 100，意思就是让画布的宽度和高度都是 100 像素。可是，我们希望标题界面与浏览器窗口大小一样，因此必须动态地覆盖这两个值，将宽度设置为 `document.body.clientWidth`，将高度设置为 `document.body.clientHeight`。

取得了 2D 上下文对象后，先调用 HTML5 Canvas API 方法 `fillStyle()`。本章一开始曾打了一个比方，把画布比喻成一张用蜡笔画有不同颜色景物的纸；这里其实也一样。在此，`fillStyle()` 方法的作用是把颜色设置为黑色，之后 `fillRect()` 方法绘制了一个填充了黑色的矩形，这个矩形的起点是 (0,0)，终点是（`canvas.width, canvas.height`），这样就让黑色矩形覆盖了整个浏览器窗口。（别忘了上一步刚刚重新设置了 `canvas` 对象的大小。）

然后再设置要使用的语句，并为文本选择了字体组合（Arial，后备字体为 sans-serif；

在 CSS 中设置也管用）及字号。接着再修改颜色——改成白色。调用 `fillText()` 方法把定义好的语句以 (10, 30) 为起点绘制到画布上（起点距画布左边 10 像素，距画布上边 30 像素）。

图 1-1 展示了执行代码后的结果。

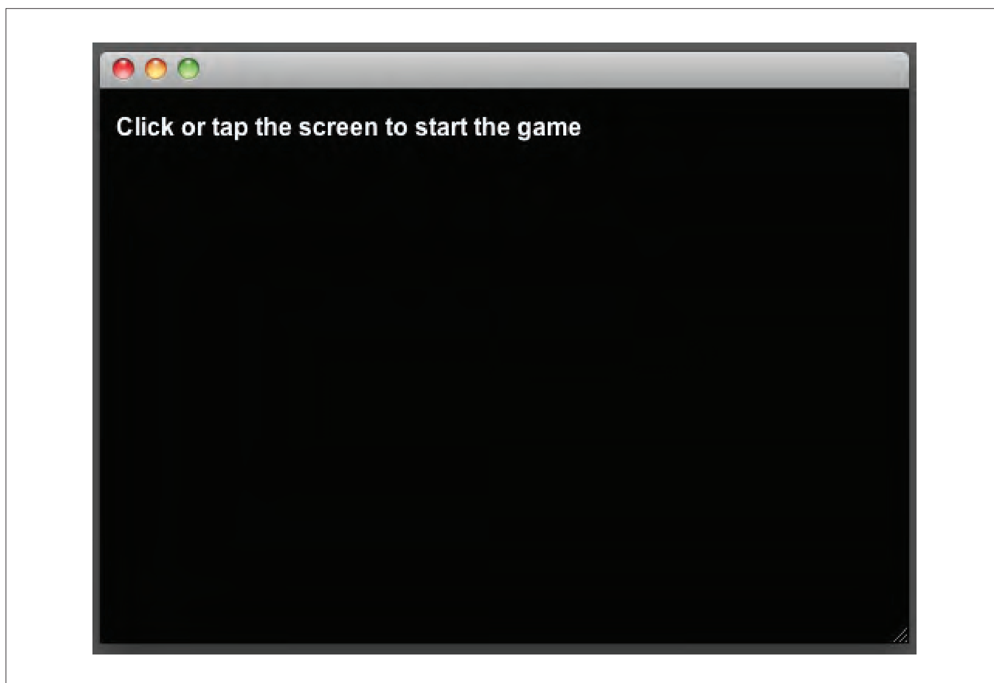


图 1-1　游戏的初始界面，使用了 canvas 元素

HTML5 Canvas API 还有一个非常有用的方法，叫做 `measureText(`*`phrase`*`)`。这个方法返回参数 *`phrase`* 的（像素）宽度。其实，还需要在设置了字号之后（而不是之前）度量一下文本的宽度。这样就可以使用 `measureText()` 把文本居中显示在屏幕中央了：

```
var phrase = "Click or tap the screen to start the game";
c.font = 'bold 16px Arial, sans-serif';
var mt = c.measureText(phrase);
var xcoord = (canvas.width / 2) - (mt.width / 2);
c.fillStyle = '#FFFFFF';
c.fillText (phrase, xcoord, 30);
```

目前，我们是把画布绘制成了黑色，只给 `fillStyle()` 方法传递了十六进制的颜色值。而 `canvas` 元素支持的其他样式还有：

- 颜色关键字，例如 'red'、'black'；
- rgb（*Red*，*Green*，*Blue*）格式的 RGB 值；
- rgba（*Red*，*Green*，*Blue*，*Alpha*）格式的 RGB 值，其中 *Alpha* 是透明度，值为 0.0 到 1.0；
- hsl（*Percentage*，*Percentage*，*Percentage*）格式的 HSL 值；
- hsla（*Percentage*，*Percentage*，*Percentage*，*Alpha*）格式的 HSLA 值。

如果实色还不够，那么 canvas 还支持：

- 使用 createLinearGradient() 方法绘制线性渐变；
- 使用 createRadialGradient() 方法绘制径向渐变；
- 使用 createPattern() 方法绘制一幅图像 / 画布或者视频。

我们可以修改前面的例子，把黑色背景换成好看的蓝色渐变。为此，可以使用下列代码：

```
var grd = c.createLinearGradient(0, 0, canvas.width, canvas.height);
grd.addColorStop(0, '#ceefff');
grd.addColorStop(1, '#52bcff');

c.fillStyle = grd;
c.fillRect(0, 0, canvas.width, canvas.height);
```

而使用 drawImage() 方法在画布上绘制图像就和绘制文本一样简单：

```
var img = new Image();
img.src = 'image.png';
c.drawImage(img, 0, 0, img.width, img.height);
```



要使用 img.width 和 img.height 属性，img.readyState 的值必须等于 COMPLETE。在最终完成的游戏代码中，我们会通过一个资源加载程序来检测这个值。这个资源加载程序在代码库中的路径及文件名是 */game/js/resourceLoader.js*。

HTML5 Canvas API 的 drawImage() 方法有 3 种实现。虽然后面几节也会涉及其他的实现形式，但要全面了解每种实现的细节，最好还是参考 W3C 的在线规范页面，地址是：http://www.w3.org/TR/2dcontext/#dom-context-2d-drawimage。

如果想让图像是原始大小的两倍，只要像下面这样把它的大小乘以 2 即可：

```
var img = new Image();
img.src = 'image.png';
c.drawImage(img, 0, 0, img.width * 2, img.height * 2);
```

在我们的例子中，要使用在线代码库的 img 目录中的 logo.png 图片。而且，我们想让这张图片填充浏览器窗口 50% 的空间，同时保持其长宽比，从而使其无论在手机、平板电脑还是桌面电脑中都能适当地显示。

为了展示标题界面，下面编写一个名为 `showIntro()` 的函数，封装显示蓝色渐变、图像及文本的代码：

```
function showIntro () {
  var phrase = "Click or tap the screen to start the game";

  // 清除画布
  c.clearRect (0, 0, canvas.width, canvas.height);

  // 绘制好看的蓝色渐变
  var grd = c.createLinearGradient(0, canvas.height, canvas.width, 0);
  grd.addColorStop(0, '#ceefff');
  grd.addColorStop(1, '#52bcff');

  c.fillStyle = grd;
  c.fillRect(0, 0, canvas.width, canvas.height);

  var logoImg = new Image();
  logoImg.src = '../img/logo.png';

  // 保存原始宽度值，以便
  // 后面保持其长宽比
  var originalWidth = logoImg.width;

  // 计算新的宽度和高度值
  logoImg.width = Math.round((50 * document.body.clientWidth) / 100);
  logoImg.height = Math.round((logoImg.width * logoImg.height) /
     originalWidth);

  // 创建一个小辅助对象
  var logo = {
      img: logoImg,
      x: (canvas.width/2) - (logoImg.width/2),
      y: (canvas.height/2) - (logoImg.height/2)
  }

  // 绘制图像
  c.drawImage(logo.img, logo.x, logo.y, logo.img.width, logo.img.height);

  // 把颜色改为黑色
  c.fillStyle = '#000000';
  c.font = 'bold 16px Arial, sans-serif';

  var textSize = c.measureText (phrase);
  var xCoord = (canvas.width / 2) - (textSize.width / 2);

  c.fillText (phrase, xCoord, (logo.y + logo.img.height) + 50);
}
```

调用 showIntro() 函数将显示如图 1-2 所示的效果。

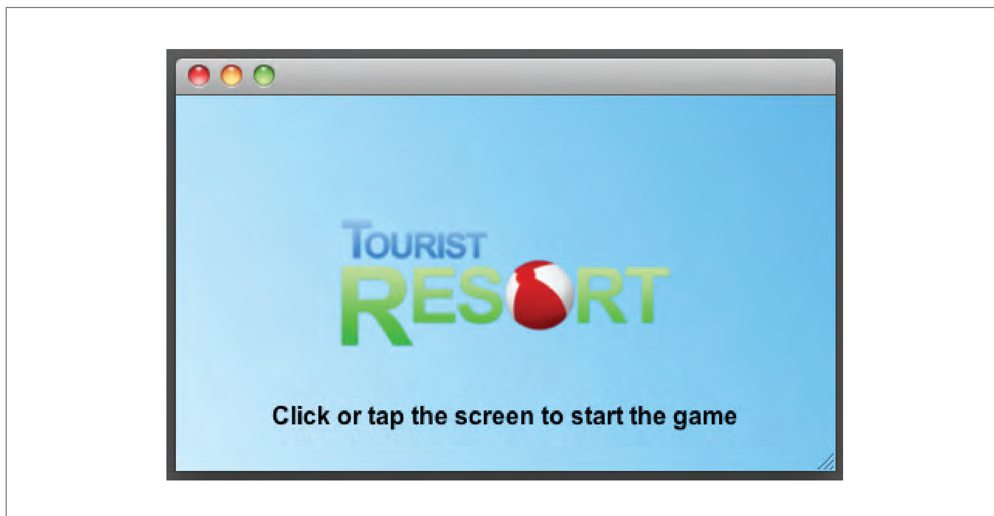


图 1-2　程序示例 1-1 的标题界面的屏幕截图

现在，我们的“标题界面”已经准备好了。下面，就该考虑怎么让这个画面淡出为白色了。要实现画面淡出为白色，可以定义一个函数，让它每 30 毫秒就调用自身一次，直到整个画面消失在白色背景中。这个函数就叫 fadeToWhite() 吧。

要是想在画布上绘制有某种透明度的区域，可以使用两种方法：

- 以 RGBA 或 HSLA 的形式指定填充颜色；
- 修改 2D 上下文中的 globaleAlpha 参数，将其设置为一个 0.0（完全透明）到 1.0（完全不透明）之间的值。

通过修改 globaleAlpha 参数（我们马上就要用这种方法），可以指定元素以多高的透明度显示在屏幕上。只要一设置了透明度，比如设置为 0.5（半透明），那么所有 fillRect()、fillText() 和 drawImage() 及其他类似调用都会得到半透明的效果。

以下就是 fadeToWhite() 函数的代码：

```
function fadeToWhite(alphaVal) {
  // 如果函数尚未接收到任何参数，则以 0.02 为起点
  var alphaVal = (alphaVal == undefined) ? 0.02 : parseFloat(alphaVal) + 0.02;

  // 将颜色设置为白色
  c.fillStyle = '#FFFFFF';
  // 设置全局透明度值
  c.globalAlpha = alphaVal;
```

```
      // 绘制与画布一样的大的矩形
      c.fillRect(0, 0, canvas.width, canvas.height);

      if (alphaVal < 1.0) {
        setTimeout(function() {
          fadeToWhite(alphaVal);
        }, 30);
      }
    }
```

到现在为止，所剩的只有附加单击和调整大小事件了。下面的程序示例 1-1 包含了完成后的代码，这些代码也可以从在线代码库中下载到，其路径和文件名是 /examples/ex1-titlescreen.html。为了节省版面，下面例子中的代码没有给出 `fadeToWhite()` 和 `showIntro()` 函数的实现，因为前面已经给出过了。

程序示例 1-1　标题界面

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>Example 1 - Title Screen</title>

    <script>
      window.onload = function () {
        var canvas = document.getElementById('myCanvas');
        var c = canvas.getContext('2d');

        var State = {
          _current: 0,
          INTRO: 0,
          LOADING: 1,
          LOADED: 2
        }

        window.addEventListener('click', handleClick, false);
        window.addEventListener('resize', doResize, false);

        doResize();

        function handleClick() {
          State._current = State.LOADING;
          fadeToWhite();
        }

        function doResize() {
          canvas.width = document.body.clientWidth;
          canvas.height = document.body.clientHeight;

          switch (State._current) {
            case State.INTRO:
                showIntro ();
                break;
```

```
      }
    }

    function fadeToWhite(alphaVal) {
      // ...
    }

    function showIntro () {
      // ...
    }
   }
  </script>
  <style type="text/css" media="screen">
    html { height: 100%; overflow: hidden }
    body {
      margin: 0px;
      padding: 0px;
      height: 100%;
    }
  </style>

 </head>
 <body>
  <canvas id="myCanvas" width="100" height="100">
    Your browser doesn't include support for the canvas tag.
  </canvas>
 </body>
</html>
```

虽然这个例子中的动画（淡出为白色）并不复杂，但如果你在移动设备运行比这个稍微复杂一点儿的动画，可能会发现动画放映过程中短暂的中断，这种现象叫做“跳帧”。

1.2 创建平滑的动画

无论开发什么游戏，最关键之处都在于如何有效地利用资源。虽然 canvas 可以非常快地在屏幕上绘制元素，但我们仍然需要对一块很大的区域每秒钟执行好几次清除和重绘。在这种情况下，桌面电脑用户可能不会感觉到画面“卡壳”，但手机和平板等移动设备就比较费劲了，而这恰恰是玩家游戏体验的重要一环。（本章后面将介绍一种显著提升性能的技术。）

不同设备之间的差异如此之大，意味着要展示一段简单的动画，那么每秒帧数（Frames Per Second，FPS）在某些设备上可能达到 90，而在有些设备却只能达到 15。

程序示例 1-2 展示了一个简单的测试，它可以告诉我们设备的大致处理能力。

程序示例 1-2　测试设备渲染 canvas 的能力

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>Canvas Example 2 (FPS Count)</title>

    <script>
      window.onload = function () {
        var canvas = document.getElementById('myCanvas');
        var c = canvas.getContext('2d');

        var fpsArray = [];
        var fpsCount = 0;
        var stopAt = 10;
        var fps = 0;
        var startTime = 0;

        var date = new Date();
        startTime = Math.round(date.getTime() / 1000);

        c.font = '20px _sans';

        draw(startTime);

        function draw (timeStamp) {
          var date = new Date();
          ts = Math.round(date.getTime() / 1000);

          if (timeStamp !== ts) {
            fps = fpsCount;
            fpsCount = 0;
            fpsArray.push(fps);
          } else {
            fpsCount++;
          }

          c.fillStyle = '#000000';
          c.fillRect (0, 0, canvas.width, canvas.height);

          c.fillStyle = '#FFFFFF';
          c.fillText ("TS: " + timeStamp, 10, 20);
          c.fillText ("FPS: " + fps, 10, 40);

          if (timeStamp <= (startTime + stopAt)) {
            setTimeout(function() {
               draw(ts);
            }, 1);
            } else {
              showResults(c, canvas);
            }
          }

          function showResults() {
```

```
            var mean = 0;
            var sum = 0;

            c.fillStyle = '#FFFFFF';
            c.fillRect (0, 0, canvas.width, canvas.height);

            // 对采样数据进行排序
            for (var i = 0; i < fpsArray.length; i++) {
              for (var j = fpsArray.length - 1; j > i; j--) {
                  if (fpsArray[j - 1] > fpsArray[j]) {
                  fpsArray[j - 1] = fpsArray[j];
                }
              }
            }

            // 丢弃第一个值，通常是非常慢的一个
            fpsArray = fpsArray.slice (1, fpsArray.length);

            for (var i = 0; i < fpsArray.length; i++) {
              sum = sum + fpsArray[i];
            }

            mean = sum / fpsArray.length;

            c.fillStyle = '#000000';
            c.fillText ("MIN: " + fpsArray[0], 10, 20);
            c.fillText ("MAX: " + fpsArray[fpsArray.length - 1], 10, 40);
            c.fillText ("MEAN: " + (Math.round(mean * 10) / 10), 10, 60);
        }
      }
    </script>

  </head>
  <body>
    <canvas id="myCanvas" width="160" height="70" style="border: 1px
        solid black;">
      Your browser doesn't include support for the canvas tag.
    </canvas>
  </body>
</html>
```

程序示例 1-2 在一秒钟内尽可能多地绘制相同的 `canvas` 对象，共绘制 10 秒。 在编写本书时，这个例子在谷歌 Chrome 和 Opera 中的性能大约是 Firefox、Safari 和 IE9 的 4 倍。Chrome 和 Opera 并不比其他浏览器更快，而是由于它们对 `setTimeout()` 和 `setInterval()` 函数人为设定了最短时间延迟。在多数浏览器中，这个值是 10 ms，而在 Chrome 和 Opera 中，这个值是 4 ms。设置这个最短延迟时间的目的，是为了避免浏览器锁定，该限制是在 W3C 的相应工作草案中规定的（参见：http://www.w3.org/TR/html5/timers.html）。

当然，对浏览器来说，更“贴心”的方式是使用 `requestAnimFrame` 函数（由于

HTML5 规范尚未最终定案，不同的浏览器可能会给这个函数起不同的名字）。使用 `requestAnimFrame` 函数可以让浏览器自己决定什么时候是显示下一帧的最佳时刻。例如，把浏览器窗口最小化，如果没有其他操作依赖于 `requestAnimFrame` 函数，那么浏览器就可以先停止动画，等窗口恢复大小并对用户可见后再重新启动它。

在本书代码库的 examples 目录中，可以看到如何使用两种方法（ex2-fpsrequest-AnimationFrame.html 和 ex2-fps-setTimeout.html）来执行这个任务（计算 FPS）的说明。有关对脚本动画进行定时控制的内容，请参见 W3C 的工作草案（见 http://webstuff.nfshost.com/anim-timing/Overview.html 和 http://www.w3.org/TR/2011/WD-animation-timing-20110602/）。ex2-fpsrequestAnimationFrame.html 中实现了一个隔离层（shim layer），检查浏览器是否实现了给定函数，如果没有则依次往后检测——相关代码的作者是 Paul Irish（http://paulirish.com）。这个隔离层是为了检测浏览器是否实现了 `requestAnimFrame()`，而将 `setTimeout()` 作为后备方法。

在后面的例子以及整个游戏开发过程中，我们还是会使用 `setTimeout()`，因为通过它可以更精确地控制何时显示下一帧。例如，我们可以每 500 ms 或 2000 ms 而不是每 1 ms 或 10 ms 调用 `setTimeout()` 函数一次，这样效率会更高。至于 `requestAnimFrame()`，虽然在本书写作时它还没有得到所有浏览器的支持，但将来在使用它时也是可以指定时间参数的。



`requestAnimFrame()` 的上限是 60 FPS。

我们已经看到了，设备的配置及性能（以及其他因素）不同，FPS 值可能会高一些，也可能会低一些。这也就意味着，如果我们的动画以帧数为基础，那么它在某些设备上播放得会更快一些。



熟悉 19 世纪 80 年代的 PC 的读者可能还记得 turbo 按钮吧，通过它可以改变处理器的时钟速度。过去，很多游戏和应用都运行在一个特定的时钟速度下，随着计算机速度越来越快，这些游戏中所有可以动的元素（包括动画）也都相应地加快了，结果显得十分滑稽。这个按钮可以让计算机“减速运行”，以便支持游戏等较早的应用。

为了避免这种不可预测的情况，我们打算在动画中采用“基于时间”的方法。这样，无论设备的处理能力是多少 FPS，都不会影响游戏效果，只要指定动画在多长时间内播完即可，从而忽略了动画实际包含的帧数。

1.3 使用精灵

为了演示这个设计思想，我们要使用 Sprite 类，这个类的作用是从一张精灵表（sprite sheet）中加载图片。而精灵就是独立的游戏贴图，其中可能包含一个（静态）或多个（动画）帧。通常，为了优化加载和内存查找的时间，我们游戏中的多数图像都将放在一个（很大的）图像文件中，这个图像文件就叫做精灵表。接下来我们要用的精灵表如图 1-3 所示。



图 1-3　一张简单的精灵表

这张精灵表总共包含 25 个图形，分为 5 组，每组 5 个。对于这张精灵表而言，它包含的就是 5 个不同的动画。我们要做的就是让第一个动画持续 5 秒，第二个动画持续 2.5 秒，第三个动画持续 1.6 秒，第四个动画持续 1.25 秒，而最后一个动画只持续 1 秒。程序示例 1-3 展示了如何实现这个效果。

程序示例 1-3　使用精灵表创建一个简单的动画

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>Canvas Example 3 (Sprite Animations)</title>
    <script charset="utf-8" src="sprite.js"></script>
    <script>
      var fpsCount = 0;
      var fps = 0;
      var startTime = 0;

      var Timer = function() {
        this.date = new Date();
      }
```

```
Timer.prototype.update = function() {
  var d = new Date();
  this.date = d;
}

Timer.prototype.getMilliseconds = function() {
  return this.date.getTime();
}

Timer.prototype.getSeconds = function() {
  return Math.round(this.date.getTime() / 1000);
}

window.onload = function() {
  var canvas = document.getElementById('myCanvas');
  var c = canvas.getContext('2d');

  //初始化精灵
  var spritesheet = '../img/sprite1.png';

  var gray = new Sprite(spritesheet, 60, 60, 0, 0, 5, 5000);
  var yellow = new Sprite(spritesheet, 60, 60, 0, 60, 5, 2500);
  var red = new Sprite(spritesheet, 60, 60, 0, 120, 5, 1666);
  var blue = new Sprite(spritesheet, 60, 60, 0, 180, 5, 1250);
  var green = new Sprite(spritesheet, 60, 60, 0, 240, 5, 1000);

  var timer = new Timer();

  c.font = '14px _sans';

  var startTime = timer.getSeconds();
  draw(startTime);

  function draw (timeStamp) {
    timer.update();

    if (timeStamp !== timer.getSeconds()) {
      fps = fpsCount;
      fpsCount = 0;
    } else {
      fpsCount++;
    }

    c.fillStyle = '#FFFFFF';
    c.fillRect (0, 0, canvas.width, canvas.height);
    c.fillStyle = '#000000';

    gray.setPosition(40, 60);
    gray.animate(c, timer);
    gray.draw(c);

    yellow.setPosition(80, 100);
    yellow.animate(c, timer);
    yellow.draw(c);
```

```
          red.setPosition(120, 140);
          red.animate(c, timer);
          red.draw(c);

          blue.setPosition(160, 180);
          blue.animate(c, timer);
          blue.draw(c);

          green.setPosition(200, 220);
          green.animate(c, timer);
          green.draw(c);

          c.fillText ("Elapsed Time: " + (timeStamp - startTime) +
              " Seconds", 10, 20);
          c.fillText ("FPS: " + fps, 10, 40);

          setTimeout(function() {
            draw(timer.getSeconds());
          }, 1);
        }
      }
    </script>

  </head>
  <body>
    <canvas id="myCanvas" width="300" height="300" style="border: 1px
        solid black;">
      Your browser doesn't include support for the canvas tag.
    </canvas>
  </body>
</html>
```

本节和其他节的例子中用到的 sprite.js 文件，其代码如下所示：

```
var Sprite = function(src, width, height, offsetX, offsetY, frames,
  duration) {
    this.spritesheet = null;
    this.offsetX = 0;
    this.offsetY = 0;
    this.width = width;
    this.height = height;
    this.frames = 1;
    this.currentFrame = 0;
    this.duration = 1;
    this.posX = 0;
    this.posY = 0;
    this.shown = true;
    this.zoomLevel = 1;
    this.setSpritesheet(src);
    this.setOffset(offsetX, offsetY);
    this.setFrames(frames);
    this.setDuration(duration);
```

```
    var d = new Date();
    if (this.duration > 0 && this.frames > 0) {
          this.ftime = d.getTime() + (this.duration / this.frames);
    } else {
          this.ftime = 0;
    }
}

Sprite.prototype.setSpritesheet = function(src) {
    if (src instanceof Image) {
          this.spritesheet = src;
    } else {
          this.spritesheet = new Image();
          this.spritesheet.src = src;
    }
}

Sprite.prototype.setPosition = function(x, y) {
    this.posX = x;
    this.posY = y;
}

Sprite.prototype.setOffset = function(x, y) {
    this.offsetX = x;
    this.offsetY = y;
}

Sprite.prototype.setFrames = function(fcount) {
    this.currentFrame = 0;
    this.frames = fcount;
}

Sprite.prototype.setDuration = function(duration) {
    this.duration = duration;
}

Sprite.prototype.animate = function(c, t) {
    if (t.getMilliseconds() > this.ftime) {
          this.nextFrame ();
    }

    this.draw(c);
}

Sprite.prototype.nextFrame = function() {
    if (this.duration > 0) {
          var d = new Date();
    if (this.duration > 0 && this.frames > 0) {
         this.ftime = d.getTime() + (this.duration / this.frames);
    } else {
         this.ftime = 0;
    }

    this.offsetX = this.width * this.currentFrame;
```

```
        if (this.currentFrame === (this.frames - 1)) {
            this.currentFrame = 0;
        } else {
            this.currentFrame++;
        }
    }
}

Sprite.prototype.draw = function(c) {
    if (this.shown) {

          c.drawImage(this.spritesheet,
                          this.offsetX,
                          this.offsetY,
                          this.width,
                          this.height,
                          this.posX,
                          this.posY,
                          this.width * this.zoomLevel,
                          this.height * this.zoomLevel);
    }
}
```

在线代码库中还有另外一个例子，在文件 ex3-spriteanim-alt.html 中，图 1-4 是这个例子的屏幕截图。

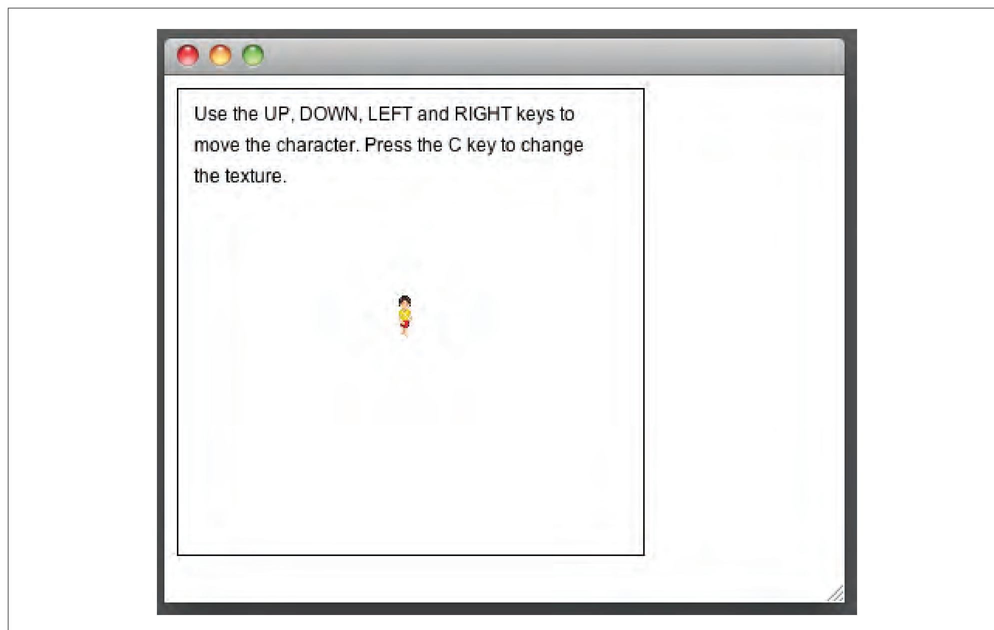


图 1-4 程序示例 1-3 的另一个版本的屏幕截图

1.4 操作像素

HTML5 Canvas 还有一个非常棒的特性，就是可以让我们访问或操作图像的每一个像素，因而就可以知道每个像素的 RGBA 值。实现像素级操作的两个方法是 `context.getImageData()` 和 `context.putImageData()`，这两个方法能够接收的参数如下：

```
context.getImageData(x, y, width, height);
```

`context.getImageData()` 返回一个 `ImageData` 对象，这个对象包含如下属性：

`width`（只读）

图像的宽度，以像素表示

`height`（只读）

图像的高度，以像素表示

`data`

一个 `CanvasPixelArray` 对象（数组），包含图像中的所有像素，每个像素占据 4 个索引（R、G、B 和 A）。如果要访问第一个像素的蓝色值，使用 `ImageData.data[2]`；要访问第二个像素的红色值，使用 `ImageData.data[4]`。

以下代码展示了访问整个画布的 `ImageData.data` 数组中所有像素值的过程：

```
var img = context.getImageData(0, 0, canvas.width, canvas.height);
var idata = img.data; // 出于性能考虑，最好是
                      // 把这个值保存在一个新数组中

for (var i = 0, idatal = idata.length; i < idatal; i += 4) {
  var red = idata[i + 0];
  var green = idata[i + 1];
  var blue = idata[i + 2];
  var alpha = idata[i + 3];
}
```

如果不是要访问图像数据，而是想向画布中插入图像数据，可以使用以下方法：

```
context.putImageData(data,x,y);
```

或者它的另一个实现：

```
context.putImageData(data,x,y,dx,dy,dw,dh);
```

这两个方法的参数说明如下：

`data`

要绘制到目标画布上的（来源）`ImageData` 对象

`x`

目标画布的 *X* 轴绘制起点（左上角为原点）

`y`

目标画布的 *Y* 轴绘制起点（左上角为原点）

[可选] `dx`（“脏 *X*”）

来源 `ImageData` 对象中的 *X* 轴起点位置

[可选] `dy`（“脏 *Y*”）

来源 `ImageData` 对象中的 *Y* 轴起点位置

[可选] `dw`（“脏宽度”）

来源 `ImageData` 对象的宽度（比如，将其指定为原始宽度的一半，会将插入的图像横向缩小 50%）

[可选] `dh`（“脏高度”）

来源 `ImageData` 对象的高度（比如，将其指定为原始高度的一半，会将插入的图像纵向缩小 50%）



为了能够在本机中运行下面的例子，需要给谷歌 Chrome、Firefox 或 Opera 传递启动参数 `--allow-file-access-from-files`，以便绕开同源策略的安全限制（每个 `file://` 都有自己的安全策略）。

要了解有关这个限制的更多信息，请参见 W3C 规范：http://dev.w3.org/html5/spec/Overview.html#security-with-canvas-elements。

Safari 没有这个限制。

程序示例 1-4 展示了如何在实际开发中使用 `getImageData()` 方法，这里是检测图像中特定像素的颜色。

程序示例 1-4　检测画布中像素的颜色

```
<!DOCTYPE html>
<html lang="en">
```

```
<head>
  <meta charset="UTF-8" />
  <title>Canvas Example 4 (Detecting Colors)</title>
<script>
  window.onload = function () {
    var preview = document.getElementById('preview');
    var r = document.getElementById('r');
    var g = document.getElementById('g');
    var b = document.getElementById('b');
    var a = document.getElementById('a');
    var mx = document.getElementById('mx');
    var my = document.getElementById('my');

    var canvas = document.getElementById('myCanvas');
    canvas.addEventListener('mousemove', move, false);

    var c = canvas.getContext('2d');

    var building = new Image()
    building.src = "../img/cinema.png";

    draw();

    function draw () {
      c.drawImage(building, 0, 0, canvas.width, canvas.height);
    }

    function move (e) {
      mx.innerHTML = e.clientX;
      my.innerHTML = e.clientY;

      var img = c.getImageData(e.clientX, e.clientY, 1, 1);
      var idata = img.data;

      var red = idata[0];
      var green = idata[1];
      var blue = idata[2];
      var alpha = idata[3];

      r.innerHTML = red;
      g.innerHTML = green;
      b.innerHTML = blue;

      a.innerHTML = (alpha > 0) ? alpha : 'Transparent';

        var rgba='rgba(' + red + ', ' + green + ', ' + blue + ', ' +
          alpha + ')';
      preview.style.backgroundColor =rgba;
    }
  }
</script>

<s tyle type="text/css" media="screen">
  body { margin: 0px; padding: 0px; }
```

```
      canvas {
        border: 1px solid black;
        float: left;
      }
      ul {
        list-style: none;
        margin: 10px 10px 10px 10px;
        padding: 0px;
        float: left;
      }

      ul li { font-weight: bold; }
      ul li span { font-weight: normal; }
      ul li #preview { width: 50px; height: 50px; border: 1px solid black; }
    </style>
  </head>
  <body>
    <canvas id="myCanvas" width="300" height="300">
      Your browser doesn't include support for the canvas tag.
    </canvas>
    <ul>
      <li><div id="preview"></div></li>
      <li>Red: <span id="r">NULL</span></li>
      <li>Green: <span id="g">NULL</span></li>
      <li>Blue: <span id="b">NULL</span></li>
      <li>Alpha: <span id="a">NULL</span></li>
      <li>Mouse X: <span id="mx">NULL</span></li>
      <li>Mouse Y: <span id="my">NULL</span></li>
    </ul>
  </body>
</html>
```

要了解更多的相关信息以及其他操作像素的方法，请访问 W3C 工作草案相关的部分：http://www.w3.org/TR/2dcontext/#pixel-manipulation。

关于透明度

学习了怎么取得画布中特定像素的颜色，包括其 alpha 值（透明度）之后，就可以解决以前 Web 开发中的一个常见问题了。之前解决这个问题要使用复杂低效的 hack（要用到 JavaScript、CSS 把 PNG 图像、`div` 或其他元素的透明区域点击一遍）。问题是这样的：虽然 PNG 图像的某些区域是透明的，但这些区域表现得仍然像是实色矩形一样。因而，点击这些透明区域会返回对原始图像的引用，而不是返回对透明区域下方的实心对象的引用。

有了前面介绍的工具，解决这个问题的途径有很多，但概括起来主要是以下两种：

- 使用 DOM 方法找到元素的位置，迭代这些元素并找到“实色”像素；
- 保存所展示的对象的位置和展示这些对象的次序，这是传统游戏开发中使用的方法。

第一种途径可以使用 DOM 的 `document.elementFromPoint()` 方法来确定单击的是什么元素，如果是一幅图像或一个有背景（包括实色背景和带有及不带有透明区域的图像背景）的对象，那么可以使用 `getImageData()` 来检测选择的像素是否透明。如果是透明的，就选择它的父元素（如果不是父元素，则可以查找同辈元素）。这样通过遍历 DOM，就可以找到那些透明的“实”色，从而选择相应的元素而非图像了。但是，这种途径如果遇到下列情形将无法运行、不够实用，非常低效：

- 目标元素有大量的同辈元素或父元素；
- 忽略了父元素的 *Z* 轴索引,穿过这个元素而选择了它下面的元素(它的一个子元素)。

第二种途径则要求我们采取与多数 Web 开发人员的习惯做法截然不同的方式：跟踪我们在屏幕上展示的对象的坐标，在有单击事件发生时进行点击检测，看看鼠标的 *X* 和 *Y* 坐标是否落在了某个对象的区域内。为此，需要保存页面中每一个元素的横、纵坐标、宽度和高度，以及将它们展现在屏幕上的次序。然后，可以通过任何方法（有很多）来遍历这些元素，检测出单击的是哪一个。在保存的每个对象横、纵坐标、宽度和高度的基础上，可以构建出相应的矩形区域，从而能够检测出鼠标事件（如 `mousedown`、`mousemove`、`mouseup`、`click` 等）返回的 `clientX` 和 `clientY` 值表示的点是否位于这些矩形区域内。图 1-5 展示了跟踪屏幕上所有对象的方式。MX 和 MY（Mouse X 和 Mouse Y）表示单击位置的坐标。然后就可以检测该坐标点是否处于某个对象（图中应该是 #4 对象）的区域中。

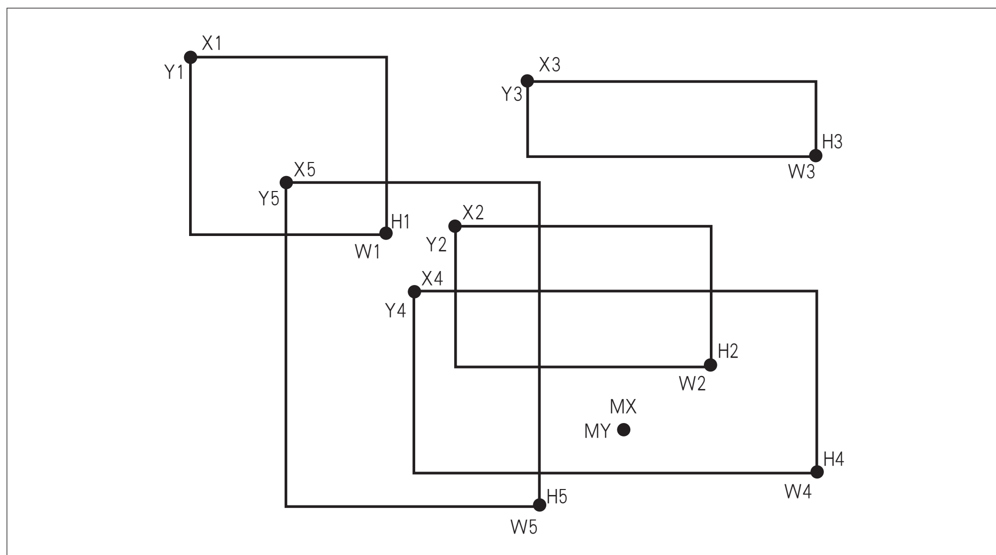


图 1-5　跟踪屏幕上的对象

在随后的例子中，我们要展示的做法能够避免使用 document.elementFromPoint() 方法的两个缺点，涉及使用 CSS 属性 pointer-events:none。这个 CSS 属性会告诉浏览器鼠标不应该与某个特定的元素交互，这样就完全避免了遍历 DOM 或跟踪屏幕上所有对象的必要。

图 1-6 展示了例子页面的组织方式。在“笑脸”<img> 的透明区域上单击就会穿透该图像，因而取得“奶酪”<div> 的引用。当然，如果单击的虽然是笑脸图像的透明区域，但点击位置正好是某个奶酪洞的上面，那么就会返回 HTML 文档的引用。

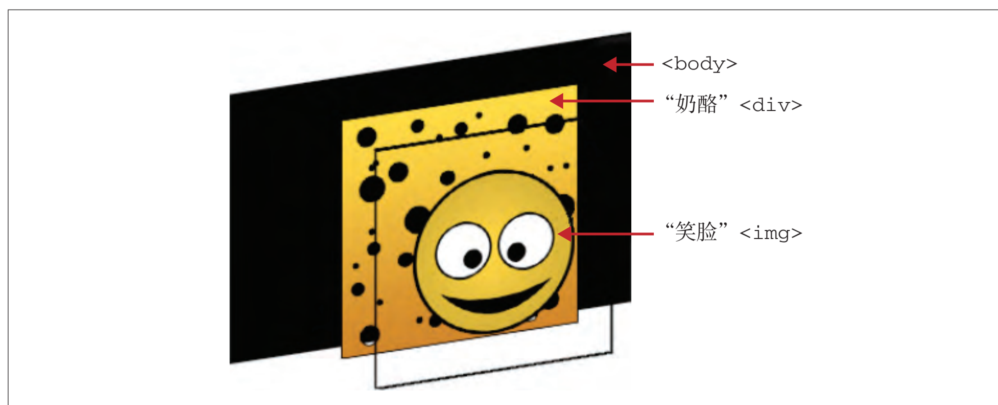


图 1-6　多层图像方式

要实现上述方法，首先要从检测页面加载完成开始：

```
window.onload = function () {
  var MIN_ALPHA_THRESHOLD = 10;

  var canvas = document.getElementById('myCanvas');
  var c = canvas.getContext('2d');

  document.addEventListener('click', detectElement, false);
```

注意有个变量叫 MIN_ALPHA_THRESHOLD，用来指定多低的透明度算是“实心”（值从 0 ~ 255，包含在 context.getImageData() 返回的像素级数据中），而非透明。所有在文档上的单击都会调用名为 detectElement() 的函数：

detectElement() 的函数的用途很简单。首先，要检测调用 document.elementFromPoint() 返回的对象，并测试其透明度。如果是透明的，则把该对象添加到一个对指针事件“不可见”的对象数组中，然后再次尝试。这样一直到找到某个实心对象或者 body，通过弹出警告框显示结果，然后回滚所有修改：

```
function detectElement (e) {
  var invisibleObjects = new Array();
  var solidPixel = false;
  var obj;

  do {
    obj = document.elementFromPoint(e.clientX, e.clientY);
    if (obj == null || obj.tagName == 'BODY' || obj.tagName == 'HTML')
      {
      break;
    }
    if (isTransparent(obj, e.clientX, e.clientY)) {
      invisibleObjects.push(obj);
      setObjectEventVisibility(obj, false);
    } else {
      solidPixel = true;
    }
  } while(!solidPixel);

    for (var i = 0; i < invisibleObjects.length; i++) {
      setObjectEventVisibility(invisibleObjects[i], true);
    }

    invisibleObjects = null;

  alert(obj.tagName);
}
```

setObjectEventVisibility() 函数用于将元素设置为对指针事件可见或不可见，它接收两个参数：元素的引用及表示是否可见的布尔值。这个函数在内部将元素的 CSS 属性 pointerEvents 设置为 visiblePainted（默认值）或 none（其他有效的值包括 visibleFill、visibleStroke、visible、painted、fill、stroke、all 和 inherit）。关于这些值的详细描述，请参考 W3C 的规范页面：http://www.w3.org/TR/SVG/interact.html#PointerEventsProperty。但我们这里只使用其中的两个值 visiblePainted 和 none。扩展这个函数支持 pointerEvents 所有值的任务，就留给读者作练习吧。

```
function setObjectEventVisibility(obj, visible) {
  if (visible) {
    obj.style.pointerEvents = 'visiblePainted';
  } else {
    obj.style.pointerEvents = 'none';
  }
}
```

用于检测图像中某个特定的坐标点是否透明的函数叫做 isPixelTransparent()。为了保证这个函数能够正确地工作，必须考虑到所有使用它的场景。例如，假设有一个 300 像素 ×300 像素的 div 元素，其背景是一个 600 像素 ×300 像素的图片，且背景水平偏移 300 像素。

图 1-7 展示了这个 div 的外观（实际上，背景图像是以“奶酪”作为纹理的），以及用作其背景的完整图片。这个 div 的背景在水平方向偏移了 300 像素。如果不考虑这个偏移量的话，单击这张图片的中央位置会得到一个透明而非实心的像素。还有一个问题要考虑（但没有体现在我们的脚本中），那就是对于非 img 元素（如 div）可以使用 CSS3 的 background-size 属性来调整背景图像与包含它的 div 的相对位置。

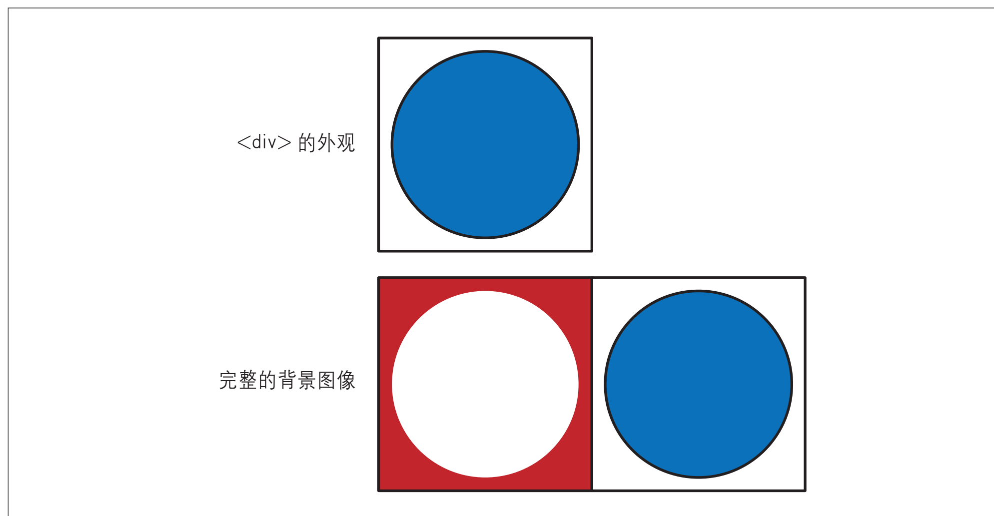


图 1-7　**div 及其背景**

因此，对于非 img 元素，我们打算使用以下辅助函数：

```
function getBackgroundPosition(src, property) {
  property = property.split(' ');
  /**
   * 如果编写检测它是否继承了父元素属性的代码，
   * 那代码的效率会很低。在此，我们假设如果它
   * 包含 'auto'，那就表示 0
   */
  var left = (property[0] != 'auto') ? property[0].substr(0, property[0].
    length - 2) : 0;
  var top = (property[1] != 'auto') ? property[1].substr(0, property[1].
    length - 2) : 0;

  return {
    x: left,
    y: top
  };
}
```

为了简单起见，我们在此假设所有背景图像不会在水平或垂直方向上重复，同时假

设所有背景都是单背景（CSS3 支持多背景）。不过，要扩展这个函数以支持多背景也很简单，只需把所有背景都加载到一个数组中即可。

也就是说，我们的 `isPixelTransparent()` 函数如下所示：

```
function isPixelTransparent (src, x, y, oWidth, oHeight, offsetX,
   offsetY) {
    var img = new Image()
    img.src = src;

    // 如果没有给这个函数传入参数，就使用“默认”值
    oWidth = (!oWidth) ? img.width : oWidth;
    oHeight = (!oHeight) ? img.height : oHeight;
    offsetX = (!offsetX) ? 0 : offsetX;
    offsetY = (!offsetY) ? 0 : offsetY;

    // 再次绘制之前，先“重置”画布
    c.clearRect(0, 0, 1, 1);

    c.drawImage(img, offsetX - x, offsetY - y, img.width, img.height);

    var idata = c.getImageData(0, 0, 1, 1);
    var data = idata.data;
    var alpha = data[3];

    return (alpha < MIN_ALPHA_THRESHOLD);
}
```

最后，`isTransparent()` 函数将负责取得由 `document.elementFromPoint()` 返回的 *X* 和 *Y* 坐标处的元素，同时在调用 `isPixelTransparent()` 之前弄清楚如何解释它。

为保证万无一失，需要先根据对象在屏幕上的位置，计算出单击的相对坐标：

```
function isTransparent(obj, x, y) {
    var robj = obj;
    var rx = robj.x;
    var ry = robj.y;
    var offset = { x: 0, y: 0 };
    var padding = { x: 0, y: 0 };
    var margin = { x: 0, y: 0 };

    // 计算相对于“最上方”父元素的 X（左）和 Y（上）坐标
    if (robj.offsetParent) {
          rx = 0;
          ry = 0;

          while(robj.offsetParent) {
               rx += robj.offsetLeft;
               ry += robj.offsetTop;
               robj = robj.offsetParent;
            }
    }
```

除了对象与其父元素之间的间距之外，它们也可能会定义内边距或外边距，因此也需要把这个因素考虑进来。图 1-8 为我们展示了一种可能的特定情况。

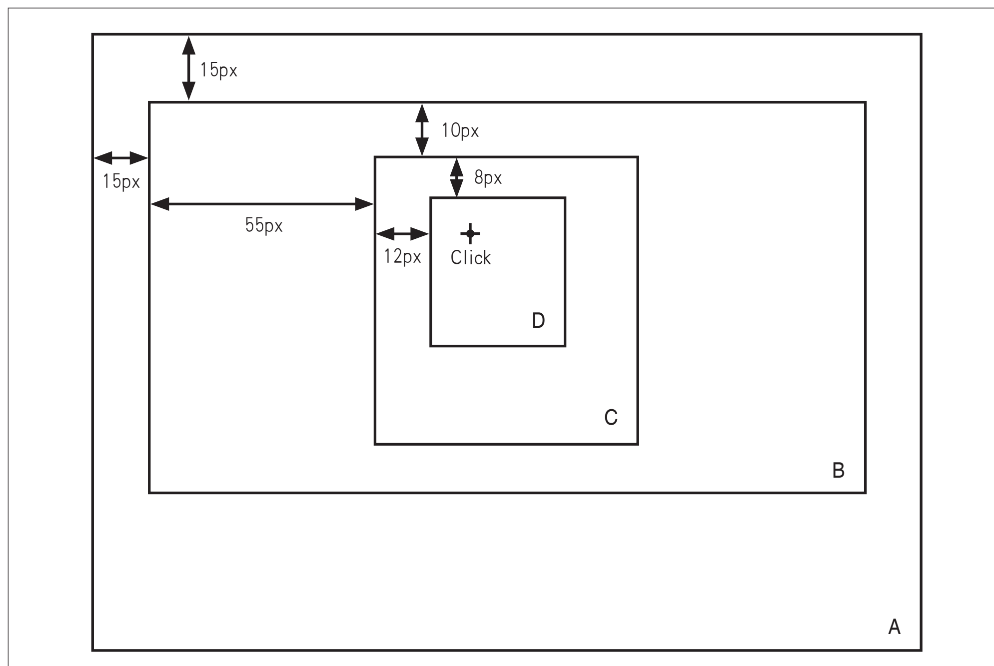


图 1-8　在一组窗口上单击

如图所示，我们检测到窗口 D 中发生的单击事件，而且我们也知道以上代码会返回相对于窗口（容器 A）边界的坐标。如果我们想知道相对于窗口 D 的 *X* 和 *Y* 坐标，那么在到达最上方（A）之前，需要通过 D 的所有其他父容器（C 和 B）。此时的计算公式如下：

```
/* 伪代码 */
xCoord = Mouse.x - 12 (C) - 55 (B) - 10 (A)
yCoord = Mouse.y - 8 (C) - 10 (B) - 15 (A)
```

容器与其所有子元素之间存在间距的原因可能有：

- 我们已经定义了子元素应该使用 `position:absolute` 或 `position:relative` 定位，设置了 `top`、`left`、`right` 或 `bottom` 值；
- 容器设置了 `padding`；
- 子元素设置了 `margin`。

以及其他原因。不过，在应用以上规则的过程中也存在一些“陷阱”。比如说，如果

父容器定义了 padding，但它的子元素使用了 position:absolute，那内边距对结果就没有什么影响。

一般来说，开发人员在设置 CSS 属性时会这样做：

```
document.getElementById('ObjectName').style.property
```

这样做的问题在于它并没有考虑通过 CSS 样式表定义的 CSS 属性，而只能在使用行内（inline）样式的情况下使用。主流浏览器大都支持 window.getComputedStyle() 方法，通过这个方法取得 CSS 属性的示例如下：

```
var cs = document.defaultView.getComputedStyle(obj, null);
paddingLeft = cs.getPropertyValue('padding-left');
```

计算的样式中的名字与 CSS 属性名相同。换句话说，访问左内边距要使用 getPropertyValue('padding-left')，而访问背景图像要使用 getPropertyValue('background-image')。

然后，需要知道我们正在处理的是什么类型的 DOM 元素。如果是图像的话，那么其处理方式肯定与 div 或 td 不一样。对于不支持背景图像或者“图像来源”的元素，应该认为它是透明的：

```
switch(obj.tagName) {
  case 'IMG':
    // 处理图像来源
    break;
  case 'DIV':
  case 'TD':
  case 'P':
  case 'SPAN':
  case 'A':
    // 处理背景图像或实心颜色
  break;
default:
  return true;
  break;
}
```

一般的 img 标签最容易处理，因为它的 src 属性中包含图像的路径：

```
case 'IMG':
  return isPixelTransparent(obj.src, (x - rx), (y - ry), obj.width, obj.height);
  break;
```

然而，对于其他元素需要一点儿技巧性才能知道它们是否有实心颜色或背景图像——如果有，那图像在该元素中如何定位和展示。

这个例子的完整代码请读者参考在线代码库 examples 目录中的 ex5-clickthrough.html 文件。

1.5 为图像选择渲染方法

据统计，网站的下载时间决定了访客的多少（因为下载速度快可以留住访客）。因此，在开发任何视频游戏的时候，都要把能够在屏幕上流畅地渲染图像和动画作为第一要务。假如帧速率过低，图像或动画不断地抖动，则必将导致大量用户流失。在等轴即时战略游戏开发中，必须时刻牢记这两点，尤其是想让自己的游戏在移动设备和桌面电脑中都能运行的情况下。

在等轴即时战略游戏开发中，游戏最基本的“目标”是在网格上面放置建筑物。每栋建筑每 N 秒钟会生成 P 个点数，从而能让玩家去买更多建筑。

虽然在 Web 浏览器（不使用任何第三方插件）中，绘制网格的最恰当、最可靠以及最高效的方式要视游戏需求而定，但有 4 种可能的手段还是可以考虑的。

- 使用 HTML5 Canvas 对象的 WebGL 上下文绘制图形，本书未作介绍（参见下面的附注）。
- 用一般的 HTML 元素（如 `div` 或 `img`）来表现图形区块（tile）和物体，使用 CSS 的 `top` 和 `left` 属性对它们进行定位。这种手段还有两种不同的手法：使用等轴图形，或者自己使用 CSS3 来旋转和斜切图形。
- 与上一种手段类似，以同样的技术包含已有元素，只不过是使用新的 CSS3 的定位工具，如 `translateX` 和 `translateY`——同时设置 `translateZ(0)`，以便在 Chrome、Safari 和 iPhone 版 Safari 中迫使硬件加速。
- 使用 HTML5 Canvas 对象的 2D 上下文。

当然还有其他渲染方法，只不过在视频游戏开发中，这些方法的效率实在是太低了。



尽管 WebGL 是在屏幕上渲染图形速度最快也最有效的方法，但我们认为它不是最可靠的。因为在本书写作时，WebGL 规范的版本号还没有达到 1.0，而且尚未得到 Internet Explorer 9、iPhone 版 Safari、Safari、Android Web 浏览器以及 Opera 的支持（Firefox for Android 支持 WebGL，但必须手工通过 `about:congif` 命令来启用）。微软已经以安全为由声明未来不会支持 WebGL（http://blogs.technet.com/b/srd/archive/2011/06/16/webgl-considered-harmful.aspx）。此外，WebGL 基于 OpenGL ES 2.0，后者是微软自己的图形库 DirectX 的竞争产品，而 WebGL 也对 Silverlight（微软的面向浏览器的 RIA 应用框架）有一定的威胁。要了解 WebGL 当前的最新信息，请参考 Khronos Group 网站的公开邮件列表：https://www.khronos.org/webgl/public-mailing-list/。

由于硬件加速的技巧（或者说功能），在本书最终上市的几个星期前，最有效的显示网格的方法是使用第三种方法。而这种方法（以及第二种方法）存在一个问题，也就是说除非我们在 DOM 树中保存大量的元素（每个元素展示一个区块，再用其他元素展示建筑物），否则必须在用户滚动屏幕时不断添加和移除节点（只显示能看得见的节点，从而保持较低的节点数）。而添加和移除节点会触发浏览器的一种非常消耗资源的操作，叫做*重排*（reflow）。重排的时候，浏览器的布局引擎会计算 DOM 树中元素的几何形状。考虑到 2D 及 3D 上下文中硬件加速功能的实现，近来（以及将来）的开发已经而且还将继续增强 HTML5 Canvas，使其成为交互式视频游戏中最合适也最有效的图形渲染方法。

等轴游戏的关键就是在网格上创建物体，而网格无非就是一个二维矩阵，我们通常所说的“*X*”轴是它的“行”，而“*Y*”轴是它的“列”。程序示例 1-5 利用 HTML5 Canvas 生成了一个 20×50 个区块（20 行 ×50 列）的矩阵。

程序示例 1-5　生成简单的网格

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>Example 6 (Generating a 20 × 50 grid)</title>
    <script>
      window.onload = function () {
        var tileMap = [];
        var grid = {
          width: 20,
          height: 50
        };

        function initializeGrid() {
          for (var i = 0; i < grid.width; i++) {
            tileMap[i] = [];
            for (var j = 0; j < grid.height; j++) {
              tileMap[i][j] = 0;
            }
          }
        }

        initializeGrid();
      }
    </script>
  </head>
  <body>
  </body>
</html>
```

在使用正确的方法的情况下，网格的大小可以不受限制，而且不会影响性能。比如可以使用 500×500 或者 1000×1000 个区块，而在到达其“开始”或“结尾”时，

利用一种内存分页程序加载和保存这些区块，而区块可以保存在计算机硬盘上（我们在此利用的是 WebStorage API），也可以保存在网络数据库服务器上。为了简单起见，我们会在游戏中使用 250×250 的网格（62 500 个区块），请读者自己动手实现内存分页功能。

现在，我们先显示正方形的网格，随后再转换到等轴视图。

假设有一个非常大的网格，包含 6 250 000 个区块（2500 行 ×2500 列），每个区块的宽度和高度都是 32 像素，而这个网格要显示在一个分辨率为 300×300 像素的容器内。很多年以来，只要是碰到与此类似的问题（每秒钟在屏幕上显示数次网格），我就会看到有人写出这样的代码：

```
for (var row = 0; row < grid.length; row++) {
  for (var col = 0; col < grid[row].length; col++) {
    displayTile(row, col);
  }
}
```

以上代码会遍历网格中每个区块的位置，到时候只管显示区块而不检查相应的区块是否已经在屏幕中了。同时，两个 `for` 循环（行和列）每迭代一次，都需要检查网格和 `grid[row]` 的大小，效率极低。

另外一些人写得比他们要讲究一点：

```
for (var row = 0, rowLength = grid.length; row < rowLength; row++) {
  for (var col = 0, colLength = grid[0].length; col < colLength;
col++) {
    if (tileIsInsideScreen(row, col)) {
      displayTile(row, col);
    }
  }
}
```

与前面的写法相比，这种写法可以把性能提升一大截。这一次，网格的大小被保存在了变量中（因而也就不必每次迭代都检查元素数目了），而且也只在区块可以显示到屏幕上的时候才会显示区块。可是，主要的问题并未解决：仍然需要 6 250 000 次迭代。想必读者一定会奇怪吧，难道有办法减少迭代的次数？算你走运，确实有。

这个办法的核心是只迭代那些能够显示的元素，而这就需要引入保存 *X*、*Y* 轴偏移（`scroll`），区块宽度、高度，以及显示区域宽度和高度的变量，例如：

```
var startRow = Math.floor(scroll.x / tile.width);
var startCol = Math.floor(scroll.y / tile.height);
var rowCount = startRow + Math.floor(canvas.width / tile.width) + 1;
var colCount = startCol + Math.floor(canvas.height / tile.height) + 1;
```

然后，就可以在循环中使用这些变量了：

```
for (var row = startRow; row < rowCount; row++) {
  for (var col = startCol; col < colCount; col++) {
    displayTile(row, col);
  }
}
```

程序示例 1-6 演示了使用这种方法创建网格的过程，图 1-9 比较了几种方法的不同结果。

程序示例 1-6　生成 2500×2500 个区块的网格

```
<!DOCTYPE html>
<html lang="en">
  <head>
  <meta charset="UTF-8" />
    <title>Example 7 (Generating a 2500 × 2500 grid)</title>
  <script>
    window.onload = function () {
      var tileMap = [];

      var tile = {
        width: 32,
        height: 32
      }

      var grid = {
        width: 2500,
        height: 2500
      }

      var Keys = {
        UP: 38,
        DOWN: 40,
        LEFT: 37,
        RIGHT: 39
      }

      var scroll = {
        x: 0,
        y: 0
      }

      var canvas = document.getElementById('myCanvas');
      var c = canvas.getContext('2d');

      window.addEventListener('keydown', handleKeyDown, false);

      initializeGrid();
      draw();

      function handleKeyDown(e) {
        switch (e.keyCode) {
          case Keys.UP:
            scroll.y -= ((scroll.y - tile.height) >= 0) ? tile.height : 0;
```

```
    break;
  case Keys.DOWN:
    scroll.y += tile.height;
    break;
  case Keys.LEFT:
    scroll.x -= ((scroll.x - tile.width) >= 0) ? tile.width : 0;
    break;
  case Keys.RIGHT:
    scroll.x += tile.width;
    break;
  }

  document.getElementById('scrollx').innerHTML = scroll.x;
  document.getElementById('scrolly').innerHTML = scroll.y;
  }
  function initializeGrid() {
    for (var i = 0; i < grid.width; i++) {
      tileMap[i] = [];
      for (var j = 0; j < grid.height; j++) {
        if ((i % 2) == 0 && (j % 2) == 0) {
          tileMap[i][j] = 0;
        } else {
          tileMap[i][j] = 1;
        }
      }
    }
  }

  function draw() {
    c.fillStyle = '#FFFFFF';
    c.fillRect (0, 0, canvas.width, canvas.height);
    c.fillStyle = '#000000';

    var startRow = Math.floor(scroll.x / tile.width);
    var startCol = Math.floor(scroll.y / tile.height);
    var rowCount = startRow + Math.floor(canvas.width / tile.
       width) + 1;
    var colCount = startCol + Math.floor(canvas.height / tile.
       height) + 1;

    for (var row = startRow; row < rowCount; row++) {
      for (var col = startCol; col < colCount; col++) {
        var tilePositionX = tile.width * row;
        var tilePositionY = tile.height * col;

        tilePositionX -= scroll.x;
        tilePositionY -= scroll.y;

        if (tileMap[row][col] == 0) {
          c.strokeRect(tilePositionX, tilePositionY, tile.width,
              tile.height);
        } else {
          c.fillRect(tilePositionX, tilePositionY, tile.width,
              tile.height);
        }
      }
    }
```

```
            setTimeout(draw, 1);
          }
        }
      </script>
    </head>
    <body>
      <canvas id="myCanvas" width="300" height="300"></canvas>
      <br/>
      Use the UP, DOWN, LEFT and RIGHT keys to scroll
      <br/>
      Scroll X: <span id="scrollx">0</span><br />
      Scroll Y: <span id="scrolly">0</span>
    </body>
  </html>
```

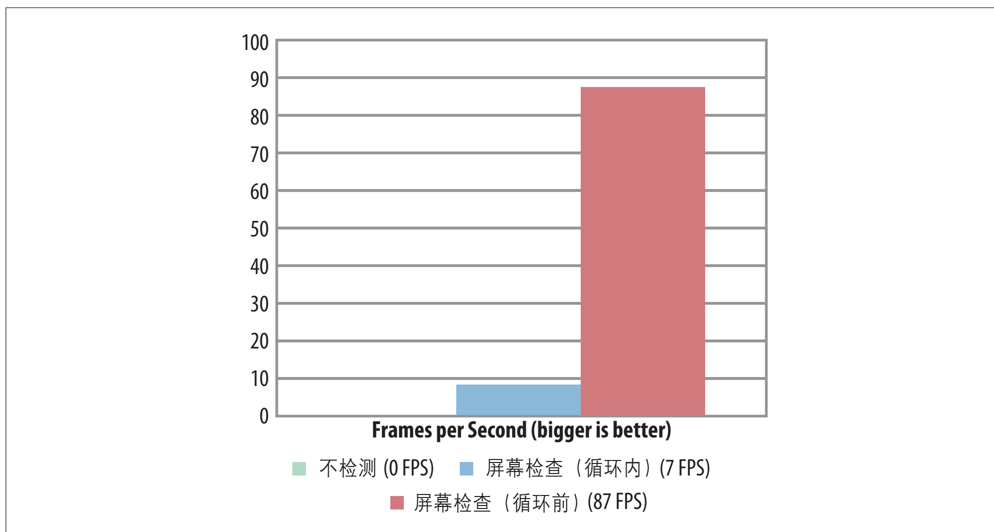


图 1-9　性能优化的结果

如图 1-9 所示，虽然性能的差异非常之大，但还有一个地方可以再优化。在前面的例子中，函数 `initializeGrid()` 负责用 1 和 0 来填充 `tileMap` 矩阵，最后在内存中保存了 6 250 000 个元素。然后，在 `draw()` 循环中，如果这个矩阵的 *X* 和 *Y* 位置值都是偶数，就绘制空心的方形；如果是奇数，就绘制实心的方形。但是，只要修改 `draw()` 中的一点儿代码，根本不用矩阵也可以达到同样的效果。就是把下面的代码：

```
if (tileMap[row][col] == 0) {
  c.strokeRect(tilePositionX, tilePositionY, tile.width, tile.height);
} else {
  c.fillRect(tilePositionX, tilePositionY, tile.width, tile.height);
}
```

改成如下所示：

```
    if ((row % 2) == 0 && (col % 2) == 0) {
      c.strokeRect(tilePositionX, tilePositionY, tile.width, tile.height);
    } else {
      c.fillRect(tilePositionX, tilePositionY, tile.width, tile.height);
    }
```

这一处小小的修改之后，初始化网格的函数就可以不要了。但现在网格的大小就没有一开始所说的 2500 × 2500 个区块的限制了。虽然这个 bug（也许会有开发人员说它是一个功能）在有的人看来还是很有用的，但如果你真想强制让网格的滚动不超过 2500 × 2500 的范围，那就还需要再在 `draw()` 函数中添加两行代码。找到下面这几行代码：

```
    var startRow = Math.floor(scroll.x / tile.width);
    var startCol = Math.floor(scroll.y / tile.height);
    var rowCount = startRow + Math.floor(canvas.width / tile.width) + 1;
    var colCount = startCol + Math.floor(canvas.height / tile.height) + 1;
```

在后面加上限制，如下所示：

```
    var startRow = Math.floor(scroll.x / tile.width);
    var startCol = Math.floor(scroll.y / tile.height);
    var rowCount = startRow + Math.floor(canvas.width / tile.width) + 1;
    var colCount = startCol + Math.floor(canvas.height / tile.height) + 1;

    rowCount = ((startRow + rowCount) > grid.width) ? grid.width : rowCount;
    colCount = ((startCol + colCount) > grid.height) ? grid.height : colCount;
```

这个经过改进之后的例子的代码，可以在代码库的 ex7-grid-canvas-alt.html 文件中找到。结果如图 1-10 所示。

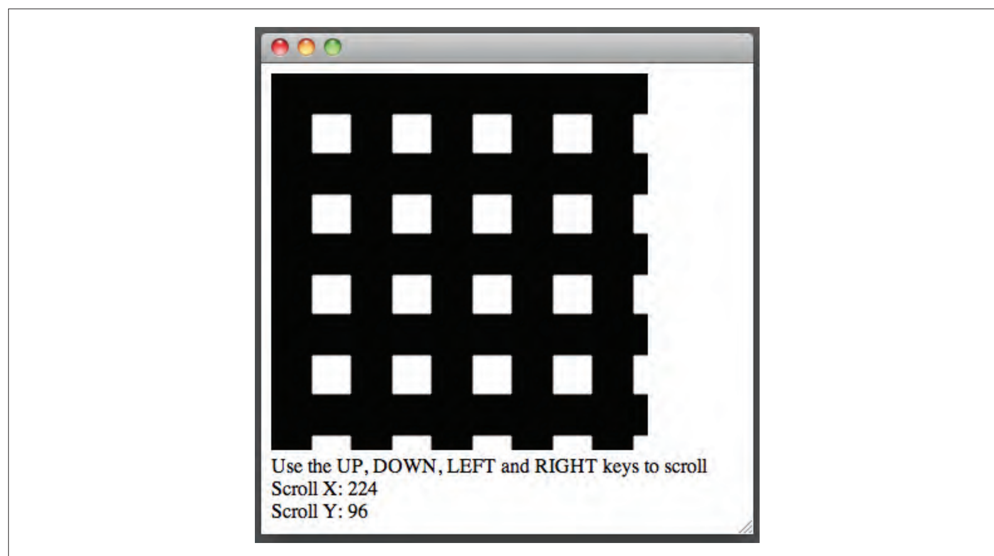


图 1-10　很多使用上下滚动视图的老式探险游戏也利用类似的手段显示地图

除了处理滚动之外，我们还可以再添加一种交互方式，即响应玩家的单击修改个别的区块。

把单击事件返回的像素坐标转换成方块网格中的矩阵坐标很简单，下面就是转换的公式：

```
var row = Math.floor(mousePositionX / tileWidth);
var column = Math.floor(mousePositionY / tileHeig);
```

至于滚动过的坐标，只要加到鼠标位置坐标上即可：

```
var row = Math.floor((mousePositionX + scrollPositionX) / tileWidth);
var column = Math.floor((mousePositionY + scrollPositionY) /
tileHeight);
```

在删除网格的初始化代码之后，出现了一个问题：怎么跟踪修改后的元素？我们运气不错，因为不必让数组的索引从 0 开始。换句话说，我们可以这样做：

```
tileMap[2423] = [];
tileMap[2423][1803] = 4;
```

这样一来，就可以只存储我们需要的元素了。而矩阵中其他没有设置值的位置将会返回 `undefined` 或 `null`。程序示例 1-7（代码库中的 ex8-grid-canvas.html）展示了实现这种新交互方式的代码，结果如图 1-11 所示。

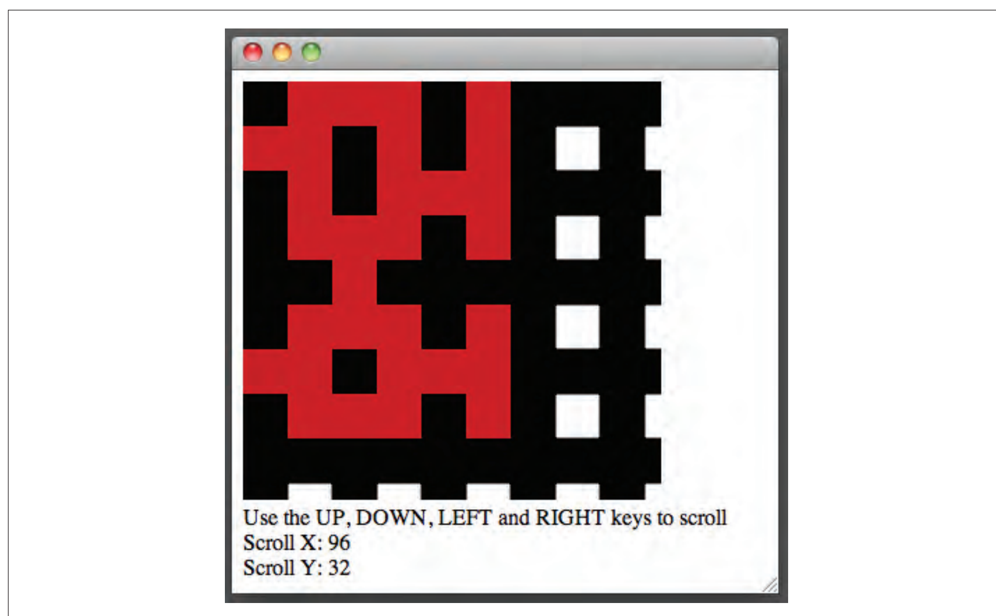


图 1-11　程序示例 1-7 的屏幕截图；红色方块表示被玩家单击的位置

程序示例 1-7　只保存需要的元素

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>Example 8</title>

    <script>
      window.onload = function () {
        var tileMap = [];

        var tile = {
          width: 32,
          height: 32
        }

        var grid = {
          width: 2500,
          height: 2500
        }

        var Keys = {
          UP: 38,
          DOWN: 40,
          LEFT: 37,
          RIGHT: 39
        }

        var scroll = {
          x: 0,
          y: 0
        }

        var canvas = document.getElementById('myCanvas');
        var c = canvas.getContext('2d');

        canvas.addEventListener('click', handleClick, false);
        window.addEventListener('keydown', handleKeyDown, false);

        draw();

        function handleClick(e) {
          // 检测到单击事件后，把鼠标的像素坐标转换成矩阵坐标
          var row = Math.floor((e.clientX + scroll.x) / tile.width);
          var column = Math.floor((e.clientY + scroll.y) / tile.height);

          if (tileMap[row] == null) {
            tileMap[row] = [];
          }
          tileMap[row][column] = 1;
        }

        function handleKeyDown(e) {
          switch (e.keyCode) {
```

```
    case Keys.UP:
      scroll.y -= ((scroll.y - tile.height) >= 0) ? tile.height : 0;
      break;
    case Keys.DOWN:
      scroll.y += tile.height;
      break;
    case Keys.LEFT:
      scroll.x -= ((scroll.x - tile.width) >= 0) ? tile.width : 0;
      break;
    case Keys.RIGHT:
      scroll.x += tile.width;
      break;
  }
  document.getElementById('scrollx').innerHTML = scroll.x;
  document.getElementById('scrolly').innerHTML = scroll.y;
}

function draw() {

  c.fillStyle = '#FFFFFF';
  c.fillRect (0, 0, canvas.width, canvas.height);
  c.fillStyle = '#000000';

  var startRow = Math.floor(scroll.x / tile.width);
  var startCol = Math.floor(scroll.y / tile.height);
  var rowCount = startRow + Math.floor(canvas.width / tile.width) + 1;
  var colCount = startCol + Math.floor(canvas.height / tile.height) + 1;

  rowCount = ((startRow + rowCount) > grid.width) ? grid.width :
     rowCount;
  colCount = ((startCol + colCount) > grid.height) ? grid.height :
     colCount;

  for (var row = startRow; row < rowCount; row++) {
    for (var col = startCol; col < colCount; col++) {
      var tilePositionX = tile.width * row;
      var tilePositionY = tile.height * col;

      tilePositionX -= scroll.x;
      tilePositionY -= scroll.y;

      if (tileMap[row] != null && tileMap[row][col] != null) {
        c.fillStyle = '#CC0000';
        c.fillRect(tilePositionX, tilePositionY, tile.width,
           tile.height);
        c.fillStyle = '#000000';
      } else {
        if ((row % 2) == 0 && (col % 2) == 0) {
          c.strokeRect(tilePositionX, tilePositionY, tile.width,
             tile.height);
        } else {
          c.fillRect(tilePositionX, tilePositionY, tile.width,
             tile.height);
        }
      }
```

```
                }
              }

              setTimeout(draw, 1);
            }
          }
        </script>
      </head>
      <body>
        <canvas id="myCanvas" width="300" height="300"></canvas>
        <br />
        Use the UP, DOWN, LEFT and RIGHT keys to scroll
        <br />
        Scroll X: <span id="scrollx">0</span><br />
        Scroll Y: <span id="scrolly">0</span>
      </body>
    </html>
```

到目前为止，我们所做的所有优化都可以显著提升游戏的性能。但在继续讨论之前，还有一件事需要考虑。

不管画布中的图形是否有变化，我们代码中的 `draw()` 循环（负责在屏幕上显示元素，是游戏的心脏和灵魂）每秒钟都会被调用很多次。对于一些屏幕上随时可能会有物体移动的动态性更强的视频游戏而言，这种做法无可厚非。但在我们关注的等轴即时战略游戏中，屏幕上的大多数物体一般都是静态的，因此这样做就有点儿不必要了。也许可以按需来调用 `draw()` 函数，而不是像现在这样每秒钟都调用它很多次。

可是不要忘了，哪怕网格有一点点儿变化，`draw()` 函数都渲染整个网格。所以，即便按需调用 `draw()` 函数，仍然存在巨大的性能损失（见图 1-12）——尤其是在只需要几个小物体移动而其他大部分场景不变的情况下。

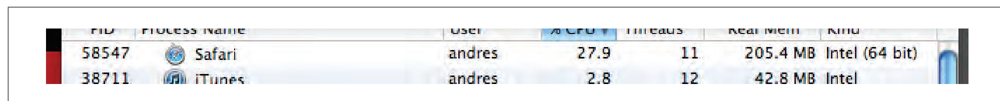


图 1-12　PC 上的 CPU 使用率约为 30%，而在移动设备上，这个指标大约在 90% ～ 100%

20 世纪 90 年代，约翰 · 卡马克（John Carmack）在 id Software 公司开发过一款游戏叫《指挥官基恩》（*Commander Keen*），这是面向 PC 平台发布的第一款滚动卷轴游戏。当时，他也遇到了跟我们现在类似的问题。为了解决这个问题，他发明了一种叫 ATR（区块适配更新）的技术，只重新绘制发生变化的区域。

为了实现与此类似的技术，我们需要去掉 `draw()` 循环中的 `setTimeout()`，为 `draw()` 函数添加 4 个参数：`srcX`、`srcY`、`destX` 和 `destY`。如果在调用 `draw()`

函数时不传递参数，则绘制整个画布；如果传递了 scrX/Y 和 destX/Y 参数，则只在给定的范围内重新绘制。

程序示例 1-8 展示了这个技术的实现。

程序示例 1-8　使用 ATR 技术修改网格

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>Example 9 - Grid modified to work with ATR (Adaptive Tile Refresh)
       </title>

    <script src="timer.js" charset="utf-8"></script>
    <script src="sprite.js" charset="utf-8"></script>
    <script>
      window.onload = function () {
        var tile = {
          width: 3,
          height: 3
        }

        var grid = {
          width: 100,
          height: 100
        }

        var canvas = document.getElementById('myCanvas');
        var c = canvas.getContext('2d');

        var man1 = new Sprite('../img/char1.png', 32, 32, 0, 96, 4, 200);
        var man2 = new Sprite('../img/char2.png', 32, 32, 0, 224, 6, 400);
        var man3 = new Sprite('../img/char3.png', 32, 32, 0, 128, 4, 600);

        var timer = new Timer();

        // 绘制整个网格
        draw();
        displayAnimatedSprites();

        function displayAnimatedSprites() {
          timer.update();

          man1.setPosition(120, 60);
          man2.setPosition(120, 102);
          man3.setPosition(120, 141);

          // 只绘制网格中变化的区域
          draw(man1.posX, man1.posY, man1.width, man1.height);
          draw(man2.posX, man2.posY, man2.width, man2.height);
          draw(man3.posX, man3.posY, man3.width, man3.height);

          man1.animate(c, timer);
          man2.animate(c, timer);
```

```
            man3.animate(c, timer);

            man1.draw(c);
            man2.draw(c);
            man3.draw(c);

            setTimeout(function() {
              displayAnimatedSprites(timer.getSeconds());
            }, 100);
          }
          function draw(srcX, srcY, destX, destY) {
            srcX = (srcX === undefined) ? 0 : srcX;
            srcY = (srcY === undefined) ? 0 : srcY;
            destX = (destX === undefined) ? canvas.width : destX;
            destY = (destY === undefined) ? canvas.height : destY;

            c.fillStyle = '#FFFFFF';
            c.fillRect (srcX, srcY, destX + 1, destY + 1);
            c.fillStyle = '#000000';

            var startRow = 0;
            var startCol = 0;
            var rowCount = startRow + Math.floor(canvas.width / tile.
              width) + 1;
            var colCount = startCol + Math.floor(canvas.height / tile.
              height) + 1;

            rowCount = ((startRow + rowCount) > grid.width) ? grid.width :
              rowCount;
            colCount = ((startCol + colCount) > grid.height) ? grid.height :
              colCount;

            for (var row = startRow; row < rowCount; row++) {
              for (var col = startCol; col < colCount; col++) {
                var tilePositionX = tile.width * row;
                var tilePositionY = tile.height * col;

                if (tilePositionX >= srcX && tilePositionY >= srcY &&
                  tilePositionX <= (srcX + destX) &&
                  tilePositionY <= (srcY + destY)) {

                  c.strokeStyle = '#CCCCCC';
                  c.strokeRect(tilePositionX, tilePositionY, tile.width,
                     tile.height);
                }
              }
            }
          }
        }
      </script>
    </head>
    <body>
      <canvas id="myCanvas" width="300" height="300"></canvas>
    </body>
  </html>
```

图 1-13 展示了应用这种技术之后，即使同时运行 3 个不同的动画，CPU 的使用率也从原来的 30% 左右明显下降到 6.5% 左右。

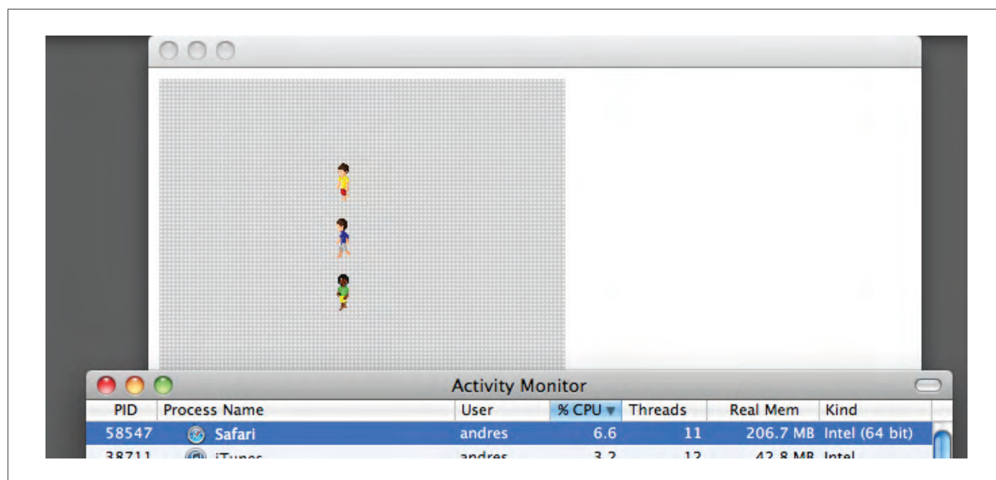


图 1-13　极大地减少了 CPU 使用率

为了简单起见，本书的例子中每次只使用一个 ATR 坐标。但在更专业、更复杂的视频游戏中，`draw()` 函数可以接收一个保存有多个 ATR 坐标的数组，从而实现一次调用即可更新多个位置的区块。比如：

```
function draw(atrArray) {
    for (var i = 0, len = atrArray.length; i < len; i++) {
          if (insideScreen(atrArray[i])) {
                drawGraphic();
          }
      }
}
```

第 2 章

理解等轴游戏



我们的游戏与其他等轴游戏，比如席德·梅尔（Sid Meier）的《文明》（*Civilization*）、Blizzard 的《暗黑破坏神》（*Diablo*），或者 Zynga 的《开心农场》（*FarmVille*）、《城市小镇》（*CityVille*）以及《咖啡世界》（*Café World*）一样，都使用了一种特殊的等轴投影（叫做正二等轴投影）视图，这种情况下的区块（通常是菱形或六边形）的宽高比是 2 ： 1。

为什么大多数游戏开发人员都选择以 2 ： 1 的比例来显示区块呢？这是由光栅图形的一个独有的问题所决定的，如果你懂得计算机显示器的工作原理，就可以理解这个问题。显示器，无论它是 CRT、TFT/LCD、LED 还是 OLED 的，都以类似我们游戏中的网格来显示像素，能够非常完美地显示垂直和水平的线条。然而，如果想要显示一条角度介于 0° 和 90° 之间的线，麻烦就来了。图 2-1 展示了这个问题。

虽然 90° 、45° 和 0° 的直线显示都很正常，而且两条平行线能够“严丝合缝”，但 30° 的直线就不行——两条线中间有缺口。但如果我们使用 2 ： 1 的宽高比，换算成角度就是 1/2 的反正切（arctan）等于 26.5650° ，能得到如图 2-2 所示的结果。

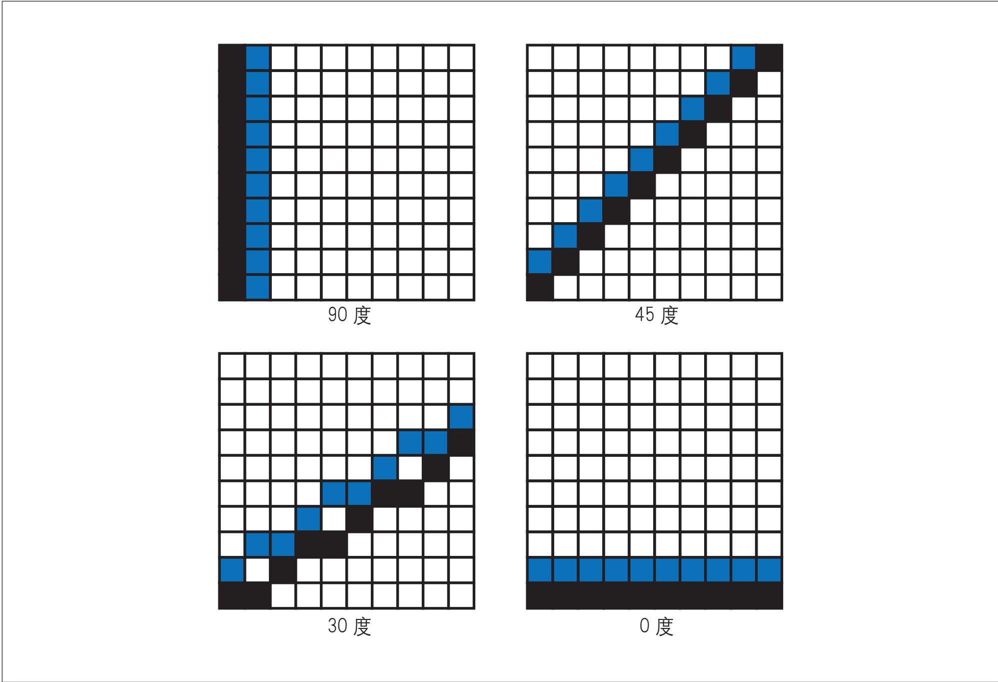


图 2-1　绘制某个角度（如 30° ）的直线会产生缺口

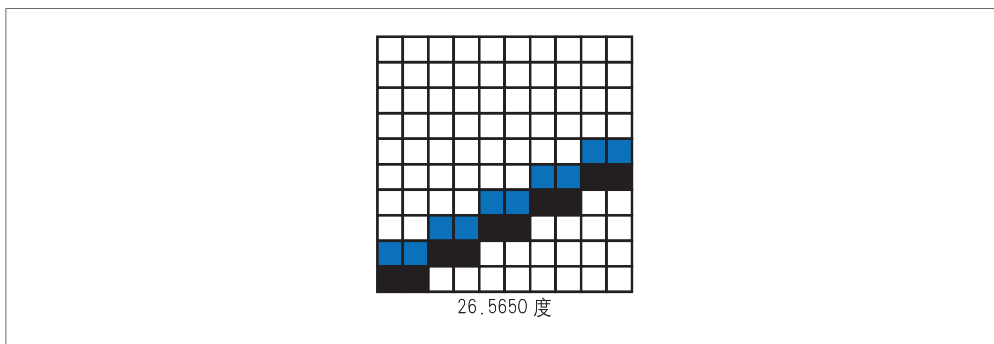


图 2-2　绘制角度为 26.5650° 的直线能得到整洁的结果

虽然到目前为止我们在网格中显示的都是正方形，但把正方形的纹理转换成等轴的纹理也很容易。程序示例 2-1 中所示的代码就可以产生图 2-3 所示的结果。

程序示例 2-1　将正方形转换成等轴菱形

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>Example 10 - (Convert square texture to isometric diamond)
       </title>

    <script>
      window.onload = function () {
        var canvas = document.getElementById('myCanvas');
        var c = canvas.getContext('2d');

        var texture = new Image();
        texture.src = '../img/squareTexture.png';

        drawDiamond();

        function drawDiamond() {
          // 保存当前的上下文
          c.save();

          // 将结果按 2 ： 1 的比例缩放成等轴菱形 / 四边形
          c.scale(1, 0.5);

          // 将上下文旋转 45°
          c.rotate(45 * Math.PI / 180);

          // 如果以图片的 (0,0) 点为圆心旋转，则一半图片
          // 会跑到画布外面，因此需要进行补偿
          c.drawImage(texture,
                        0,
                        0,
                        texture.width,
```

```
                        texture.height,
                        texture.width / 2,
                        (texture.height / 2) * -1,
                        texture.width,
                        texture.height);

                // 恢复上下文
                c.restore();
            }
        }
    </script>
  </head>
  <body>
    <canvas id="myCanvas" width="300" height="300"></canvas>
  </body>
</html>
```

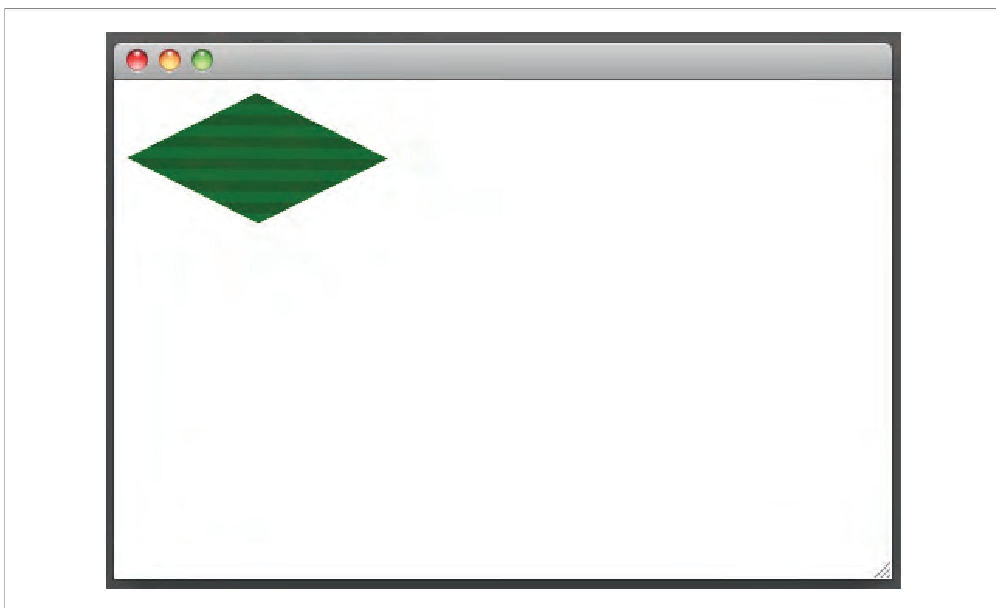


图 2-3　程序示例 2-1 的输出结果

如果把这个 drawDiamond() 函数与前一章讨论的 HTML5 Canvas getImageData() 和 putImageData() 函数组合起来，那么可以先保存变换的结果，将来再重用它以便只通过一次变换来显示多个元素。

问题在于使用 putImageData() 函数的速度比调用 drawImage() 的速度慢。因此更有效的手段就是直接使用菱形的图片（可能负责贴图制作的艺术家会稍微麻烦一点）。

理解了等轴区块的排布规则之后，在等轴网格中绘制和操作就并不困难。图 2-4 展示了等轴区块的排列规则。

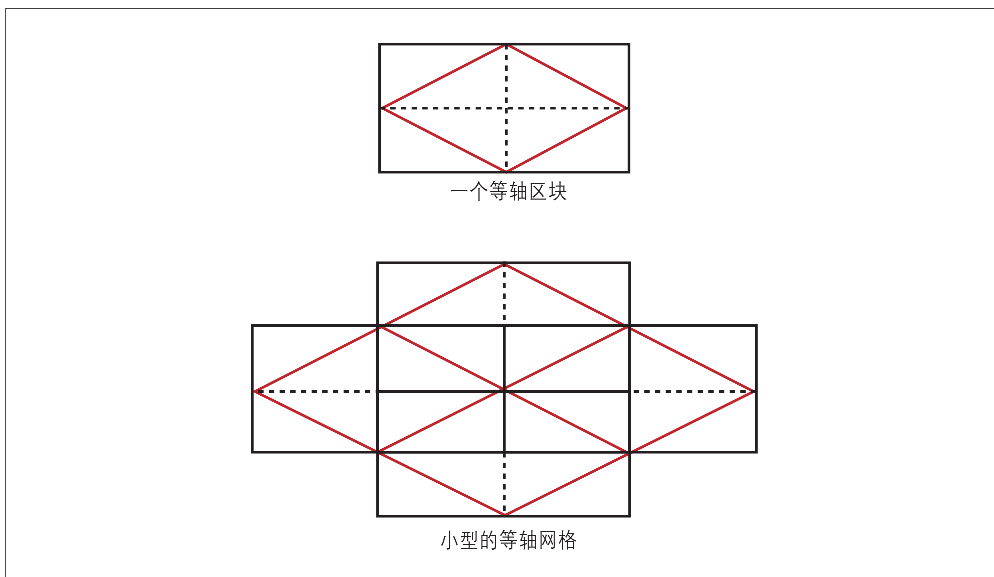


图 2-4　在区块基础上构建等轴网格

如图所示，一个等轴区块不过就是宽等高两倍的矩形。这个矩形中的菱形通常是一张图片或矢量图形，可以通过以下坐标来确定：

左

X：`0`

Y：`tile.height/2`

右

X：`tile.width`

Y：`tile.height/2`

上

X：`tile.width/2`

Y：`0`

下

X：`tile.width/2`

Y：`tile.height`

这就意味着，如果我们想显示两个区块：

(1) 首先需要定位第一个区块；

(2) 第二个区块应该放在第一个区块的宽度除以 2 和高度除以 2 的位置上。

同样，考虑到这个放置区块的计算过程，还要确保区块的宽度和高度必须是偶数（可以被 2 整除）；否则，就无法构建协调的网格了。

程序示例 2-2 展示了如何显示基本的等轴网格。稍后，我们会再修改这个例子，利用另一种高性能的手段。

程序示例 2-2　显示等轴网格的一种简单但效率不高的方式

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>Example 11 - (Displaying an isometric grid)</title>
  <script>
    window.onload = function () {
      var canvas = document.getElementById('myCanvas');
      var c = canvas.getContext('2d');

      var tile = new Image();
      tile.src = "../img/tile.png";

      draw();

      function draw() {

        c.clearRect (0, 0, canvas.width, canvas.height);

        for (var col = 0; col < 10; col++) {
          for (var row = 0; row < 10; row++) {
            var tilePositionX = (row - col) * tile.height;

            // 水平居中网格
            tilePositionX += (canvas.width / 2) - (tile.width / 2);

            var tilePositionY = (row + col) * (tile.height / 2);

            c.drawImage(tile,
                        Math.round(tilePositionX),
                        Math.round(tilePositionY),
                        tile.width,
                        tile.height);
            }
          }
        }
      }
    </script>
  </head>
  <body>
    <canvas id="myCanvas" width="600" height="300"></canvas>
  </body>
</html>
```

在等轴网格中将点击坐标转换为矩阵坐标要比在方块网格中复杂，而在处理无限大网格的时候更是这样。为方便和性能起见，我们的例子（以及最终的游戏）都将使用固定的 250 × 250（62 500 个区块）网格，这样已经比同类型的其他游戏大了。

确定计算公式之前，不能忘了把水平居中网格时导致的偏移考虑进来（参见图 2-6）。

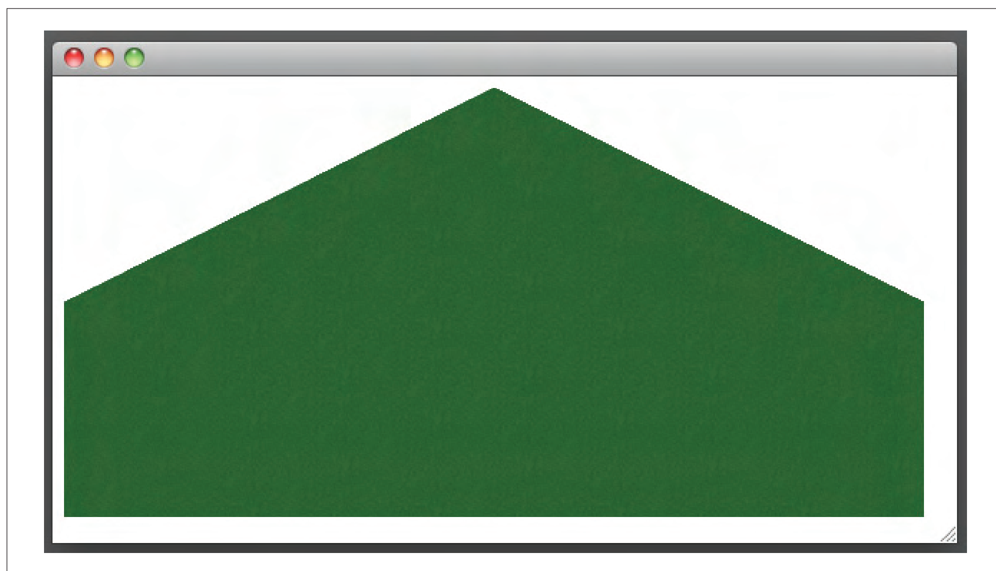


图 2-5　程序示例 2-2 的输出结果

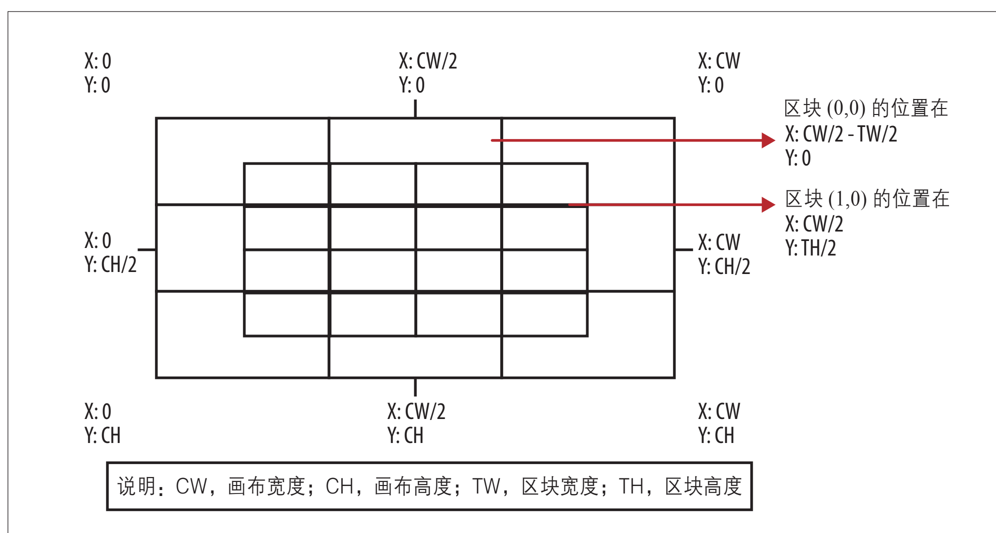


图 2-6　每个区块是怎样在画布上面定位的，以及怎样计算点击了哪个区块

首先，需要考虑 *X* 轴或 *Y* 轴的偏移量，比如滚动位置或水平 / 垂直方向的对齐。只要有一处偏移未考虑到，在变换坐标时都会导致问题。

```
var gridOffsetY = grid.height;
var gridOffsetX = grid.width;

// 考虑由于水平居中网格导致的 x 轴的偏移
gridOffsetX += (canvas.width / 2) - (tile.width / 2);
```

使用这两个变量，就可以变换列的坐标了：

```
var col = (e.clientY - gridOffsetY) * 2;
col = ((gridOffsetX + col) - e.clientX) / 2;
```

计算出列之后，对行也可以如法炮制：

```
var row = ((e.clientX + col) - tile.height) - gridOffsetX;
```

最后，用得到的结果除以区块的高度，然后再四舍五入为整数：

```
row = Math.round(row / tile.height);
col = Math.round(col / tile.height);
```

程序示例 2-3 是在实际应用中使用以上公式的例子，例子中通过放置区块创建了图 2-7 中所示的图像。

程序示例 2-3　把点击坐标转换成矩阵坐标

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>Example 12 - (Capturing click events and translating them
            to matrix coordinates)</title>

    <script>
      window.onload = function () {
        var grid = {
          width: 10,
          height: 10
        }

        var canvas = document.getElementById('myCanvas');
        var c = canvas.getContext('2d');

        var tileMap = [];

        var tile = new Image();
        tile.src = "../img/tile.png";

        var dirt = new Image();
        dirt.src = "../img/dirt.png";
```

```
canvas.addEventListener('mousedown', handleMouseDown, false);
draw();

function handleMouseDown(e) {
  var gridOffsetY = 0;
  var gridOffsetX = 0;

  // 考虑由于水平居中网格导致的 X 轴的偏移
  gridOffsetX += (canvas.width / 2) - (tile.width / 2);

  var col = (e.clientY - gridOffsetY) * 2;
  col = ((gridOffsetX + col) - e.clientX) / 2;

  var row = ((e.clientX + col) - tile.height) - gridOffsetX;

  row = Math.round(row / tile.height);
  col = Math.round(col / tile.height);

  // 检查边界
  if (row >= 0 &&
    col >= 0 &&
    row <= grid.width &&
    col <= grid.height) {

    tileMap[row] = (tileMap[row] === undefined) ? [] : tileMap[row];

    tileMap[row][col] = 1;
    draw();
  }
}

function draw() {

  c.clearRect (0, 0, canvas.width, canvas.height);

  for (var col = 0; col < grid.height; col++) {
    for (var row = 0; row < grid.width; row++) {

    var tilePositionX = (row - col) * tile.height;

    // 水平居中网格
    tilePositionX += (canvas.width / 2) - (tile.width / 2);

    var tilePositionY = (row + col) * (tile.height / 2);

    if (tileMap[row] != null && tileMap[row][col] != null) {
      c.drawImage(dirt,
                  Math.round(tilePositionX),
                  Math.round(tilePositionY),
                  dirt.width,
                  dirt.height);
  } else {
    c.drawImage(tile, Math.round(tilePositionX), Math.round
      (tilePositionY),tile.width, tile.height);
    }
  }
}
```

```
        }
      }
    </script>
  </head>
  <body>
    <canvas id="myCanvas" width="600" height="300"></canvas>
  </body>
</html>
```

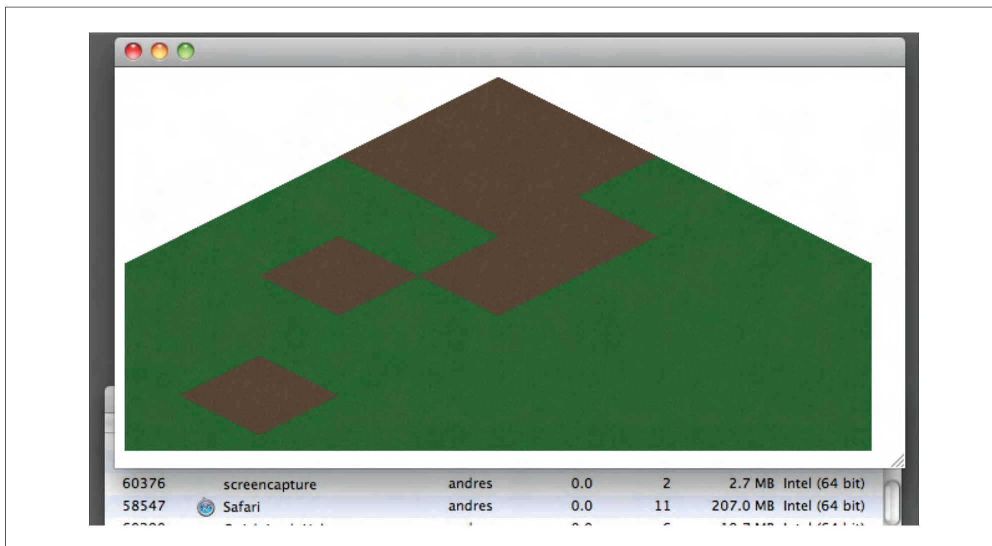


图 2-7　使用新的区块显示方法，CPU 使用率为 0%

本书在线代码库中有一个示例文件叫 ex12-isogrid-click-alt.html，它展示了如何实现矩阵的旋转。

知道了如何在等轴网格中将点击坐标转换成矩阵坐标之后，那下面就该想一想怎么放置建筑物了，如图 2-8 所示。



图 2-8　占据 4 个区块位置的建筑物贴图

图 2-8 以及图 2-9 中的建筑物贴图高度并不一致，而且要占据多个区块位置，因此需要使用不同的方法来放置它们，这一点与一般的区块是不同的。

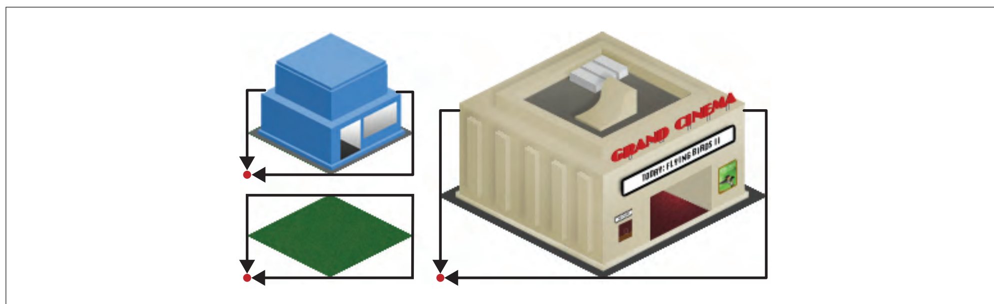


图 2-9　将建筑物放置在区块的底边中心点

如图 2-9 所示，为了正确地显示建筑物，必须找到区块的底边中心点坐标，代码如下：

```
for (var row = 0; row < 10; row++) {
  for (var col = 0; col < 10; col++) {

    var tilePositionX = (row - col) * tile.height;

    // 水平居中网格
    tilePositionX += (canvas.width / 2) - (tile.width / 2);

    var tilePositionY = (row + col) * (tile.height / 2);

    if (tileMap[row] != null && tileMap[row][col] != null) {
      tilePositionY -= building.height - tile.height;
      tilePositionX -= (building.width / 2) - (tile.width / 2);
      c.drawImage(building,
                  Math.round(tilePositionX),
                  Math.round(tilePositionY),
                  building.width,
                  building.height);
    } else {
      c.drawImage(tile, Math.round(tilePositionX), Math.round(tilePositionY),
        tile.width, tile.height);
    }
  }
}
```

在线代码库中的 ex13-isogrid-buildings.html 文件包含完整的例子。

到目前为止，我们只是使用网格来保存某个对象是否可见：只跟踪了哪些区块被占用，而没有跟踪这些区块的位置上都放置了什么物体。显然，要开发游戏必须还得设计更复杂的对象结构，特别是需要放置的建筑物占据多个区块的情况下，例如电影院（2×2 个区块）或酒店（2×2 个区块）。

为了防止用户把一个建筑物放在另一个建筑物上面，我们要使用的方法就是检查用户点击的区块是否包含某种特殊对象（叫做 `BuildingPortion`），该对象中保存着其中包含“真实”建筑物对象的主区块的坐标。图 2-10 展示了这个概念。

	1	2	3	4
1		objId: 2 x: -1 y: 0		Building objId: 1 **type: C**
2			objId: 4 x: 0 y: -1	objId: 4 x: -1 y: -1
3	objId: 3 x: 0 y: -1	objId: 3 x: -1 y: -1	Building objId: 4 **type: A**	objId: 4 x: -1 y: 0
4	Building objId: 3 **type: A**	objId: 3 x: -1 y: 0		

图 2-10　计算建筑物的位置

在图 2-10 中：

- 类型为 A（例如 `objId = 3` 或 `4`）的建筑物的大小为 2×2 个区块；
- 类型为 B（`objId = 2`）的建筑物的大小为 2×1；
- 类型为 C（`objId = 1`）的建筑物的大小为 1×1。

假设我们在网格中查询位置 (3,2)，就会发现如果向左向下移动一个区块，就会碰到 `objId = 3` 的建筑物。这些位置都是在放置每个建筑物的时候自动创建的。

程序示例 2-4（ex13-isogrid-buildings-alt.html）展示了如何实现这个想法，图 2-11 展示了结果。

程序示例 2-4　实现建筑物位置追踪

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>Example 13 - (Placing buildings)</title>

    <script>
      var Cinema = function(instanceId) {
        this.buildingTypeId = 1; // 是一个电影院
        this.instanceId = null;
```

```
  this.texture = new Image();
  this.texture.src = "../img/cinema.png";

  this.width = this.texture.width;
  this.height = this.texture.height;

  this.tileWidth = 2;
  this.tileHeight = 2;
}

var BuildingPortion = function(buildingTypeId, x, y) {
  this.buildingTypeId = buildingTypeId;
  this.x = x;
  this.y = y;
}

window.onload = function () {
  var grid = {
    width: 10,
    height: 10
  }

  var canvas = document.getElementById('myCanvas');
  var c = canvas.getContext('2d');

  var tileMap = [];
  var tile = new Image();
  tile.src = "../img/tile.png";

  var buildingCounter = 0; // 实际应用中，建筑物的计数在服务器端通过数据库完成

  canvas.addEventListener('mousedown', handleMouseDown, false);

  draw();

  function handleMouseDown(e) {
    var gridOffsetY = 0;
    var gridOffsetX = 0;

    // 考虑由于水平居中网格导致的 x 轴的偏移
    gridOffsetX += (canvas.width / 2) - (tile.width / 2);

    var col = (e.clientY - gridOffsetY) * 2;
    col = ((gridOffsetX + col) - e.clientX) / 2;

    var row = ((e.clientX + col) - tile.height) - gridOffsetX;

    row = Math.round(row / tile.height);
    col = Math.round(col / tile.height);

    // 创建建筑物对象
    var cinema = new Cinema(buildingCounter);

    // 检测边界
    if (row >= 0 &&
      col >= 0 &&
```

```
row <= grid.width &&
col <= grid.height) {

tileMap[row] = (tileMap[row] === undefined) ? [] : tileMap[row];

// 网格中有没有足够的空间放置这个建筑物?
if (((row+1) - cinema.tileWidth) < 0 || ((col+1) - cinema.
  tileHeight) < 0) {
alert("Invalid Location!\nPart of the building will appear
  outside the grid.");
  return;
}

// 检查尚未被其他建筑物占用而当前建筑物将要占用的区块
for (var i = (row+1) - cinema.tileWidth; i <= row; i++) {
  for (var j = (col+1) - cinema.tileHeight; j <= col; j++) {
    if (tileMap[i] != undefined && tileMap[i][j] != null) {
      alert("There's another building there!")
      return;
    }
  }
}
// 放置建筑物
for (var i = (row+1) - cinema.tileWidth; i <= row; i++) {
  for (var j = (col+1) - cinema.tileHeight; j <= col; j++) {
    tileMap[i] = (tileMap[i] == undefined) ? [] : tileMap[i];

    tileMap[i][j] = (i == row && j == col) ? // 强制换行
                    cinema : // 强制换行
                    new BuildingPortion(cinema.buildingTypeId,
                     i, j);
    }
  }

  buildingCounter++;

  draw();
  }
}

function draw() {

  c.clearRect (0, 0, canvas.width, canvas.height);

  for (var col = 0; col < 10; col++) {
    for (var row = 0; row < 10; row++) {

      var tilePositionX = (row - col) * tile.height;

      // 水平居中网格
      tilePositionX += (canvas.width / 2) - (tile.width / 2);

      var tilePositionY = (row + col) * (tile.height / 2);

      if (tileMap[row] != null && tileMap[row][col] != null) {
```

```
                    tilePositionY -= tileMap[row][col].height - tile.height;
                    tilePositionX -= (tileMap[row][col].width / 2) - (tile.
                        width / 2);

                    if (!(tileMap[row][col] instanceof BuildingPortion)) {
                        c.drawImage(tileMap[row][col].texture,
                                    Math.round(tilePositionX),
                                    Math.round(tilePositionY),
                                    tileMap[row][col].width,
                                    tileMap[row][col].height);
                    }
                } else {
                c.drawImage(tile, Math.round(tilePositionX), Math.
                    round(tilePositionY),tile.width, tile.height);
                }
            }
        }
    }
}
</script>
</head>
<body>
  <canvas id="myCanvas" width="600" height="300"></canvas>
</body>
</html>
```



图 2-11　程序示例 2-4 的屏幕截图

与前几节使用的常规网格不同，要在等轴网格中只显示那些我们需要的区块，需要多做些工作以及更多的迭代。图 2-12 展示了如果我们向上滚动，列的迭代从 1 而不是 0 开始，就会有一组区块显示不出来，因为这些区块同时依赖于列和行的迭代。

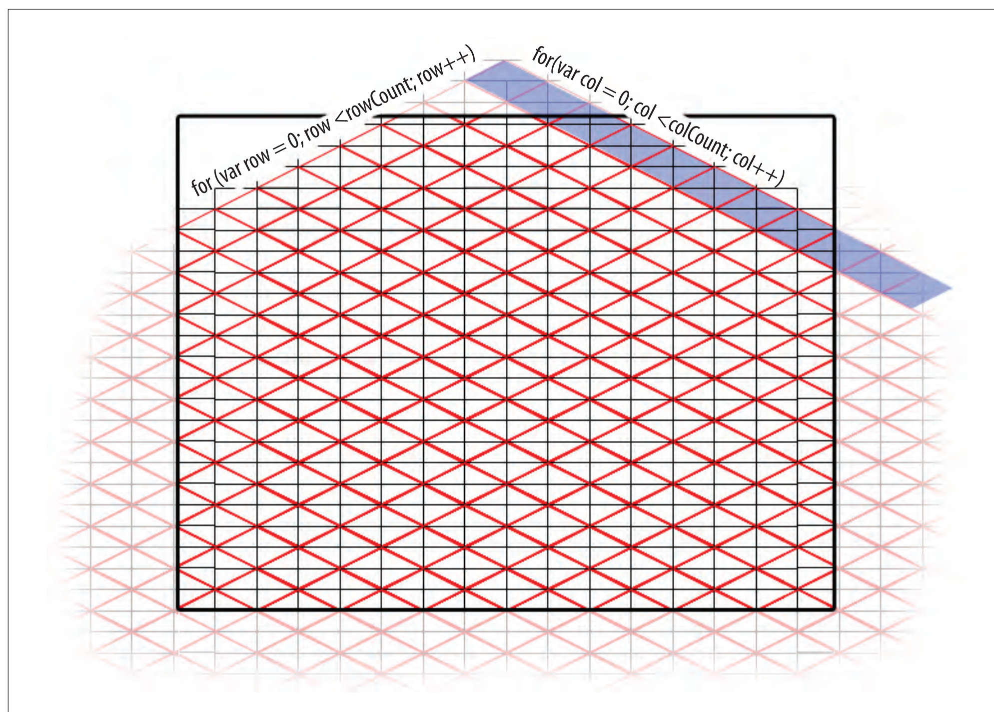


图 2-12 即使 (0,0) 位置的区块在屏幕外面，仍然需要遍历第 0 列（及第 0 行）；否则，就会有其他（应该显示的）区块显示不出来

有一个简单的办法可以解决这个问题，那就是再次使用我们的单击检测代码，找到显示在左上角、右上角、左下角和右下角的区块：

```
var pos_TL = translatePixelsToMatrix(canvas, tile, 1, 1);
var pos_BL = translatePixelsToMatrix(canvas, tile, 1, canvas.height);
var pos_TR = translatePixelsToMatrix(canvas, tile, canvas.width, 1);
var pos_BR = translatePixelsToMatrix(canvas, tile, canvas.width,
   canvas.height);

var startRow = pos_TL.row;
var startCol = pos_TR.col;
var rowCount = pos_BR.row + 1;
var colCount = pos_BL.col + 1;

startRow = (startRow < 0) ? 0 : startRow;
startCol = (startCol < 0) ? 0 : startCol;
```

```
// 放置人为的限制
rowCount = (rowCount > grid.width) ? grid.width : rowCount;
colCount = (colCount > grid.height) ? grid.height : colCount;
for (var row = startRow; row < rowCount; row++) {
    for (var col = startCol; col < colCount; col++) {

          // ...
    }
}
```

下面是 translatePixelsToMatrix() 函数的代码：

```
function translatePixelsToMatrix(canvas, tile, x, y) {
  var gridOffsetY = (grid.height * zoomHelper.level) + scrollPosition.y;
  var gridOffsetX = (grid.width * zoomHelper.level);

  // 默认情况下，网格是水平居中的
  gridOffsetX += (canvas.width / 2) - ((tile.width / 2) * zoomHelper.level) +
scrollPosition.x;

  var col = (2 * (y - gridOffsetY) - x + gridOffsetX) / 2;
  var row = x + col - gridOffsetX - tile.height;

  col = Math.round(col / tile.height);
  row = Math.round(row / tile.height);
  return {
    row: row,
    col: col
  }
}
```

在接下来的几章以及最终的游戏中，我们还将继续使用这个改进后的办法。

第 3 章

游戏界面设计



要构建一个外观漂亮的游戏，远不止在网格上放置建筑物那么简单。真正的游戏要提供多种方式供用户与游戏交互，而在使用移动设备玩游戏时，这种交互方式又会不一样。而用户对移动设备中的游戏和对桌面游戏的期望也是不一样的。换句话说，移动设备为游戏界面设计增添了新的复杂性。

3.1 Web游戏中的GUI设计和交互

GUI（Graphical User Interface，图形用户界面）和 HCI（Human-Computer Interaction，人机交互）是应用开发中极其重要的一部分，但往往被开发人员所忽略。Apple、Google 以及 Facebook 等公司和应用之所以能够成功，很大一部分原因就是它们遵循了卓越的可用性设计指南。一致、简洁、快速以及最小化设计体现在它们的每一款产品中。当然，视频游戏（也包括我们的）设计也不例外，不过这里边有一个很大的问题需要注意：与其他产品不同，我们的 GUI 必须同时适应桌面及移动设备的屏幕。

明确了这一点之后，我们可以有针对性地列举出一组建议。

- 不能依赖鼠标悬停给出反馈或显示提示条。触摸屏只有两个状态：触摸与未触摸。
- 没有提示条，就只能通过图标和按钮来清晰地传达通过它们能做什么的信息。
- 也不能依赖于右键单击，因为移动设备没有那个功能。
- 在某些等轴游戏中，横向游戏模式比纵向游戏模式能更有效地利用屏幕区域。
- 账户余额等重要信息最好显示在顶部，而导航元素最好显示在左下角。（别忘了全世界有 90% 的人是右利手，在使用右手来点击游戏界面的同时，他们通常都用左手拿着设备，如图 3-1 所示。）

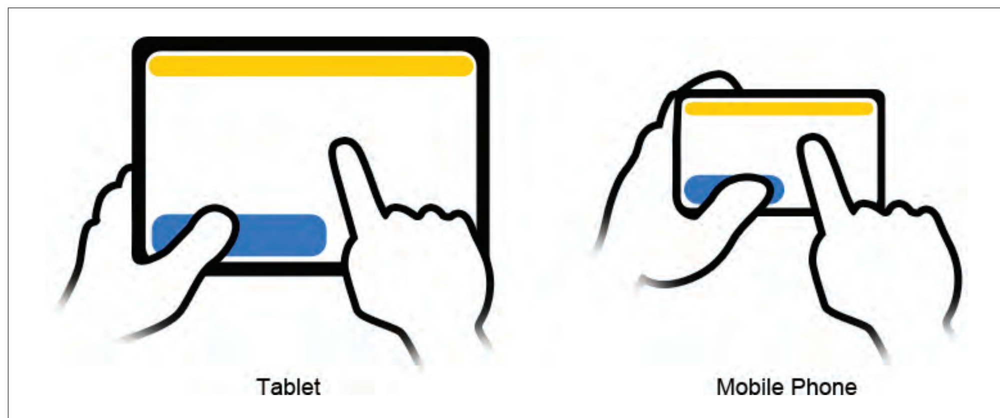


图 3-1　绝大多数人用左手拿着设备

- 由于手机屏幕很小，应该尽量保持界面的简洁。
- 应该假设用电脑鼠标上下滚动与使用两根手指放大缩小（目前的标准做法）执行相同的操作。
- 还应该避免显示浮动窗口，而代之以滚动面板。这些面板会在屏幕的右侧显示。



以下是两个针对移动设备的 HCI 和 UE（User Experience，用户体验）设计指南：

- 苹果公司的 iOS Human Interface Guidelines：http://t.cn/hb5g1R[1]
- 谷歌公司（针对 Android 设备）的 User Experience Guidelines：http://t.cn/hSRKa[2]

大体上，我们游戏的 GUI 类似图 3-2 所示。

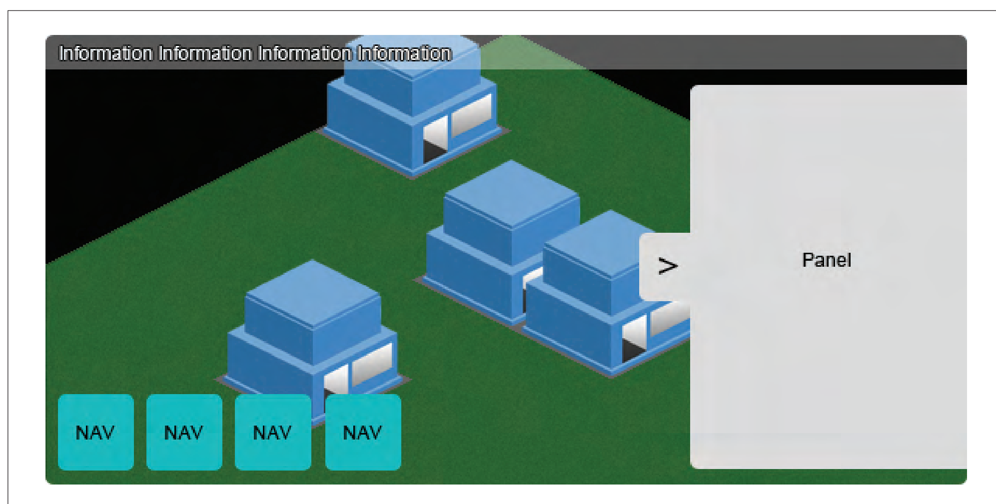


图 3-2　游戏界面轮廓

每个导航按钮都可以调出一种工具，或者执行一种特定的操作。我们的游戏将会添加下列工具。

- 选择工具，用于选择不同的建筑物。
- 移动工具，用于切换滚动位置修改器，可以通过鼠标使用；在移动设备或平板电脑中可以通过手指在屏幕上拖动使用。
- 放大及缩小按钮，用于放大网格及其中的物体，操作方法是点击屏幕。

注 1：长 URL：http://developer.apple.com/library/ios/#documentation/UserExperience/Conceptual/MobileHIG/Introduction/Introduction.html。（译者注）

注 2：长 URL：http://developer.android.com/guide/practices/ui_guidelines/index.html。（译者注）

- 旋转按钮，用于逆时针旋转矩阵的值。
- 拆除按钮，用于拆掉特定的建筑物。

此外，我们还要支持以下功能。

- 不通过缩放按钮，直接使用鼠标的滚轮放大或缩小。
- 使用键盘上的键来滚动网格或者放大、缩小以及旋转网格。

3.2 实现GUI

页面 HTML 代码的 <head> 标签中也要包含一些特殊的标签。

因为我们要以不同的方式来处理缩放，不能使用移动浏览器中默认的缩放机制，所以必须完全禁用这个默认的功能：

```
<meta name="viewport" content="width=device-width, initial-scale=1,
   user-scalable= no"/>
```

苹果 iPhone、iPad 以及部分 Android 手机允许用户将我们的站点添加到他们的主屏幕上。为此，可以使用下列标签修改我们的应用图标：

```
<link rel="apple-touch-icon" href="../img/touristResortIcon.png" />
<link rel="apple-touch-icon-precomposed" href="../img/
   touristResortIcon.png"/>
```

如果用户已经把应用添加到了主屏幕上，那么可以使用以下标签来隐藏浏览器默认的 GUI 部件（如地址栏、后退 / 前进按钮等）：

```
<meta name="apple-mobile-web-app-capable" content="yes" />
```

最后，还需要添加 Google Chrome 浏览器内嵌框架[3]，以便在老版本的浏览器中显示我们的游戏：

```
<meta http-equiv="X-UA-Compatible" content="chrome=1" />
```

游戏的 GUI 将包含在一个 div 元素中，它的 id 值为 ui。这样，就可以通过类似如下的代码来监听这个元素及其子元素中发生的任何事件：

```
var ui = document.getElementById('ui');

// 监听 GUI 事件
ui.addEventListener('mouseup', handler, false);
```

而 GUI 的 HTML 代码如下所示：

注 3：参见 http://code.google.com/intl/zh-CN/chrome/chromeframe/。（译者注）

```
<div id="ui">
  <div id="top">
      Account Balance: <span id="balance">0</span> Coins
  </div>
  <div id="tools">
    <ul>
      <li id="select"></li>
      <li id="move"></li>
      <li id="zoomIn"></li>
      <li id="zoomOut"></li>
      <li id="rotate"></li>
      <li id="demolish"></li>
    </ul>
  </div>
  <div id="panel-container" class="hidden">
    <a href="javascript:void(0)" id="panel-toggle">Build</a>
    <div id="panel">
      <h3>Choose a building:</h3>
      <ul id="buildings">
        <li>
          <h2>Building Name</h2>
          <p>
            Description
            <br />
            <span>$Cost</span>
          </p>
        </li>

        <li>
          <h2>Building Name</h2>
          <p>
            Description
            <br />
            <span>$Cost</span>
          </p>
        </li>

        <li>
          <h2>Building Name</h2>
          <p>
            Description
            <br />
            <span>$Cost</span>
          </p>
        </li>
      </ul>
    </div>
  </div>
</div>
```

处理界面中单击事件的函数如下所示：

```
var ui = document.getElementById('ui');
```

```
// 监听 GUI 事件
ui.addEventListener('mouseup', function(e) {
  switch(e.target.getAttribute('id')) {
    case 'panel-toggle':
      // ...
      break;
    case 'select':
      // ...
      break;
    case 'move':
      // ...
      break;
    case 'zoomIn':
      // ...
      break;
      case 'zoomOut':
        // ...
        break;
      case 'rotate':
        // ...
        break;
      case 'demolish':
        // ...
        break;
  }
}, false);
```

前面的面板（panel-container）及账户余额（balance）代码中包含一些虚构的值，注意到了吗？没关系，稍后我们会通过由服务器端脚本生成的客户端脚本向其中填入真实的值。

移动设备能够检测 click、mouseDown、mouseMove、mouseUp 和 DOMMouseScroll 事件，但（不同的硬件）强加了从 250 ~ 600 毫秒的限制。为了解决这个问题，需要检测设备是否支持触摸事件，如果支持则使用原生的触摸和手势事件，例如 touchstart、touchmove 和 touchend；检测手势要使用 gesturestart 和 gestureend。

可以使用 Modernizr 库来检测并管理单击和触摸事件。如果设备支持触摸事件，则 Modernizr.touch 返回 true，而这意味着我们可以这样做：

```
var pointer = {
  DOWN: 'mousedown',
  UP: 'mouseup',
  MOVE: 'mousemove'
};

if (Modernizr.touch){
  pointer.DOWN = 'touchstart';
  pointer.UP = 'touchend';
  pointer.MOVE = 'touchmove';
}
```

```
window.addEventListener('resize', function() { doResize(canvas); }, false);
canvas.addEventListener(pointer.DOWN, handleMouseDown, false);
canvas.addEventListener(pointer.MOVE, handleDrag, false);
document.body.addEventListener(pointer.UP, handleMouseUp, false);

if (Modernizr.touch){
  // 检测手势
  document.body.addEventListener('gestureend', handleGestureEnd, false);

} else {
  document.body.addEventListener('keydown', handleKeyDown, false);

  // 检测滚动
  document.body.addEventListener('mousewheel', handleScroll, false);
  document.body.addEventListener('DOMMouseScroll', handleScroll, false);

}
```

随着基础代码越来越多，为了简化工作，我们还要创建一个 Game 类；此外，还要把页面中的代码分割成多个脚本。

程序示例 3-1 展示了本书在线代码库中 ex14-gui.html 的完整代码，运行结果参见图 3-3。代码库中的 ex14-gui-sound.html 也包含相同的示例，只不过配上了背景音乐。

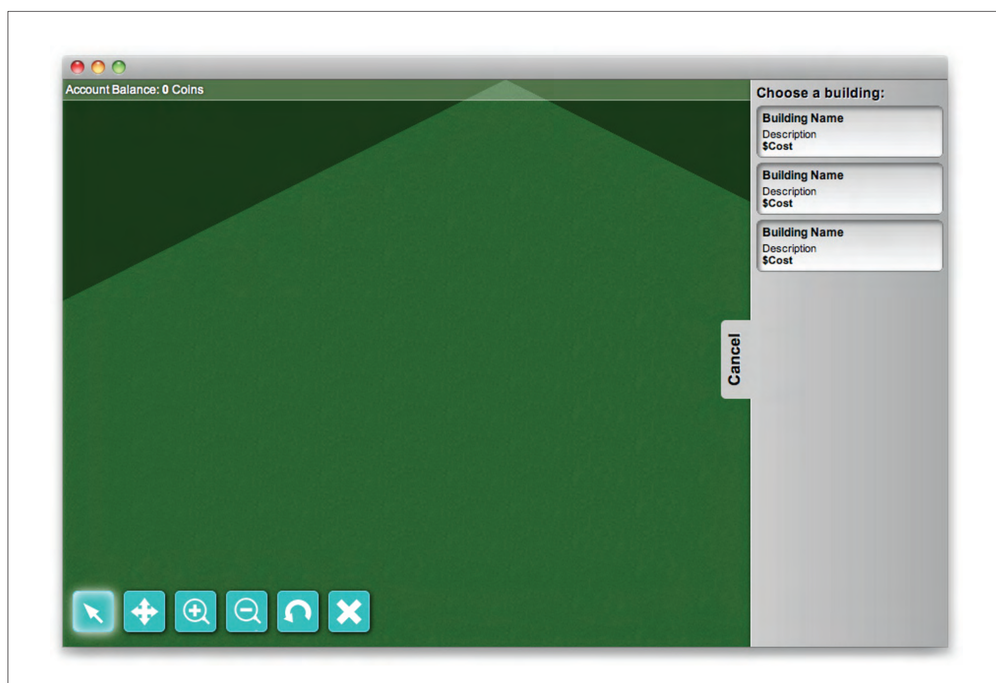


图 3-3　程序示例 3-1 运行结果的屏幕截图

程序示例 3-1　游戏的 GUI（HTML）

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />

    <!-- 我们需要自己来处理缩放 -->
    <meta name="viewport" content="width=device-width,initial-scale=1,
       user-scalable=no" />

    <!-- iPhone 图标及无默认部件的浏览器 -->
    <meta name="apple-mobile-web-app-capable" content="yes" />

    <!-- iPhone 主屏幕图标 -->
    <link rel="apple-touch-icon" href="../img/touristResortIcon.png" />
    <link rel="apple-touch-icon-precomposed" href="../img/
       touristResortIcon.png"/>

    <!-- Chrome 浏览器内嵌框架 -->
    <meta http-equiv="X-UA-Compatible" content="chrome=1" />

    <title>Example 14 - Graphical User Interface</title>

    <link rel="stylesheet" href="ui-style.css" />
    <script src="../utils/modernizr-1.7.min.js" charset="utf-8"></script>
    <script src="game-ex14.js" charset="utf-8"></script>
    <script>
    // 枚举
    var Keys = {
      UP: 38,
      DOWN: 40,
      LEFT: 37,
      RIGHT: 39,
      W: 87,
      A: 65,
      S: 83,
      D: 68,
      Z: 90,
      X: 88,
      R: 82
    }

    var Tools = {
      current: 4, // 默认工具
      /* - */
      MOVE: 0,
      ZOOM_IN: 1,
      ZOOM_OUT: 2,
      DEMOLISH: 3,
      SELECT: 4,
      BUILD: 5
    }

    window.onload = function () {
```

```
var canvas = document.getElementById('gameCanvas');
var game = document.getElementById('game');

// 初始化游戏对象
var g = new Game(canvas, game, 500, 500);

var pointer = {
  DOWN: 'mousedown',
  UP: 'mouseup',
  MOVE: 'mousemove'
};

if (Modernizr.touch){
  pointer.DOWN = 'touchstart';
  pointer.UP = 'touchend';
  pointer.MOVE = 'touchmove';
}

// 设置事件监听器
window.addEventListener('resize', function() {
       g.doResize();
  }, false);
canvas.addEventListener(pointer.DOWN, function(e) {
       g.handleMouseDown(e);
  }, false);
canvas.addEventListener(pointer.MOVE, function(e) {
       g.handleDrag(e);
  }, false);
document.body.addEventListener(pointer.UP, function(e) {
       g.handleMouseUp(e);
  }, false);

if (Modernizr.touch){
  // 检测手势
  document.body.addEventListener('gestureend', function(e)
     { g.handleGestureEnd(e); }, false);

  } else {
    document.body.addEventListener('keydown', function(e) {
        g.handleKeyDown(e); },false);
  // 检测滚轮滚动
  document.body.addEventListener('mousewheel', function(e) {
     g.handleScroll(e); },false);
  document.body.addEventListener('DOMMouseScroll', function(e)
     { g.handleScroll(e); }, false);
}

// 监听 GUI 事件
var ui = document.getElementById('ui');
ui.addEventListener(pointer.UP, function(e) {
  switch(e.target.getAttribute('id')) {
    case 'panel-toggle':
      var panelContainer = document.getElementById('panel-container');
      var classes = panelContainer.getAttribute('class');
```

```
      if (classes != null && classes.length > 0) {
        panelContainer.setAttribute('class', '');
        document.getElementById('panel-toggle').innerHTML = 'Cancel';
      } else {
        panelContainer.setAttribute('class', 'hidden');
        document.getElementById('panel-toggle').innerHTML = 'Build';
      }
      break;
    case 'select':
      selectTool(Tools.SELECT, document.getElementById('select'));
      break;
    case 'move':
      selectTool(Tools.MOVE, document.getElementById('move'));
      break;
    case 'zoomIn':
      selectTool(Tools.ZOOM_IN, document.getElementById('zoomIn'));
      break;
    case 'zoomOut':
      selectTool(Tools.ZOOM_OUT, document.getElementById('zoomOut'));
      break;
    case 'rotate':
      g.rotateGrid();
      g.draw();
      break;
    case 'demolish':
      selectTool(Tools.DEMOLISH, document.getElementById('demolish'));
      break;
    default:
      // 他没有单击任何选项，实际上只单击了 UI 中的空白区域
      // 因此将画布作为来源元素
      e.srcElement = canvas;
      e.target = canvas;
      e.toElement = canvas;

      g.handleMouseDown(e);

      break;
  }
 }, false);
}

function selectTool(tool, elem) {

  // 删除 div#tools ul 中所有的 active 类
  for (var i = 0, x = elem.parentNode.childNodes.length; i < x; i++) {
    if (elem.parentNode.childNodes[i].tagName == "LI") {
      elem.parentNode.childNodes[i].className = null;
    }
  }

  elem.className += "active";

  switch(tool) {
```

```
        case Tools.SELECT:
          Tools.current = Tools.SELECT;
          break;
        case Tools.MOVE:
          Tools.current = Tools.MOVE;
          break;
        case Tools.ZOOM_IN:
          Tools.current = Tools.ZOOM_IN;
          break;
        case Tools.ZOOM_OUT:
          Tools.current = Tools.ZOOM_OUT;
          break;
        case Tools.DEMOLISH:
          Tools.current = Tools.DEMOLISH;
          break;
      }

    }
  </script>
  </head>
  <body>
    <div id="game">

    <canvas id="gameCanvas" width="1" height="1"></canvas>
    <div id="ui">
    <div id="top">
      Account Balance: <span id="balance">0</span> Coins
    </div>
    <div id="tools">
      <ul>
        <li id="select" class="active"></li>
        <li id="move"></li>
        <li id="zoomIn"></li>
        <li id="zoomOut"></li>
        <li id="rotate"></li>
        <li id="demolish"></li>
      </ul>
    </div>
    <div id="panel-container" class="hidden">
      <a href="javascript:void(0)" id="panel-toggle">Build</a>
      <div id="panel">
        <h3>Choose a building:</h3>
        <ul id="buildings">
          <li>
            <h2>Building Name</h2>
              <p>
                Description
                <br />
                <span>$Cost</span>
              </p>
            </li>

            <li>
              <h2>Building Name</h2>
              <p>
```

```
                Description
                <br />
                <span>$Cost</span>
              </p>
            </li>

            <li>
              <h2>Building Name</h2>
              <p>
                Description
                <br />
                <span>$Cost</span>
              </p>
            </li>
          </ul>
        </div>
      </div>
    </div>

  </div>
  </body>
</html>
```

程序示例 3-2　实现游戏功能的代码（JavaScript）

```
// 例 14 中的 Game 类
function Game(canvas, game, gridSizeW, gridSizeH) {
    this.started = true;
    this.gameContainer = game;
    this.canvas = canvas;

    // 取得 2D 上下文对象
    this.c = canvas.getContext('2d');

    // 现在可以运行游戏吗？
    var missingDeps = [];
    var dependencies = [Modernizr.rgba,
                            Modernizr.canvas,
                            Modernizr.borderradius,
                            Modernizr.boxshadow,
                            Modernizr.cssgradients];
    for (var i = 0, dep = dependencies.length; i < dep; i++) {
          if (!dependencies[i]) {
                missingDeps.push(dependencies[i]);
          }
    }

    if (missingDeps.length !== 0) {
          var msg = "This browser doesn't include some of the ";
          msg += "technologies needed to play the game";
          alert(msg);
          this.started = false;
          return;
    }
```

```
    // 区块的贴图
    this.tile = new Image();
    this.tile.src = "../img/tile.png";

    // 网格的大小
    this.grid = {
          width: gridSizeW,
          height: gridSizeH
    }

    // 区块地图矩阵
    this.tileMap = [];

    // 拖动辅助对象
    this.dragHelper = {
          active: false,
          x: 0,
          y: 0
    }

    // 缩放辅助对象，支持三级缩放
    this.zoomHelper = {
          level: 1,
          NORMAL: 1,
          FAR: 0.50,
          CLOSE: 2
    }

    // 滚动位置辅助对象，跟踪滚动坐标
    this.scrollPosition = { x: 0, y: 0 }

    // 默认的缩放级别
    this.tile.width *= this.zoomHelper.level;
    this.tile.height *= this.zoomHelper.level;

    // 初始时，水平及垂直居中起点位置
    var nspy = (this.grid.height * this.zoomHelper.level) + this.
       scrollPosition.y;
    var nspx = (this.grid.width * this.zoomHelper.level) + this.
       scrollPosition.x;
    this.scrollPosition.y -= nspy;
    this.scrollPosition.x -= nspx;

    this.doResize();
    this.draw();
}

Game.prototype.handleGestureEnd = function(e) {
    e.preventDefault();

    if (Math.floor(e.scale) == 0) {
          this.zoomIn();
    } else {
          this.zoomOut();
    }
```

```
}

Game.prototype.handleScroll = function(e) {
    e.preventDefault();

    var scrollValue = (e.wheelDelta == undefined) ? e.detail * -1 :
       e.wheelDelta;

    if (scrollValue >= 0) {
          this.zoomInt();
    } else {
          this.zoomOut();
    }
}

Game.prototype.handleKeyDown = function(e) {
    switch (e.keyCode) {
          case Keys.UP:
          case Keys.W:
               this.scrollPosition.y += 20;
               break;
          case Keys.DOWN:
          case Keys.S:
               this.scrollPosition.y -= 20;
               break;
          case Keys.LEFT:
          case Keys.A:
               this.scrollPosition.x += 20;
               break;
          case Keys.RIGHT:
          case Keys.D:
               this.scrollPosition.x -= 20;
               break;
          case Keys.X:
               this.zoomIn();
               break;
          case Keys.Z:
               this.zoomOut();
               break;
          case Keys.R:
               this.rotateGrid();
               break;
    }
    this.draw();
}

Game.prototype.handleDrag = function(e) {
          var x, y;
          e.preventDefault();

          if (Modernizr.touch) {
                  x = e.touches[0].pageX;
                  y = e.touches[0].pageY;
          } else {
                  x = e.clientX;
```

```
                y = e.clientY;
        }

        switch (Tools.current) {
                case Tools.MOVE:
                        if (this.dragHelper.active) {
                                // 平滑滚动效果
                                this.scrollPosition.x -= (this.
                                   dragHelper.x - x) / 18;
                                this.scrollPosition.y -= (this.
                                   dragHelper.y - x) /18;
                        }
                        this.draw();
                        break;
        }
}

Game.prototype.handleMouseUp = function(e) {
    e.preventDefault();

    switch (Tools.current) {
          case Tools.MOVE:
               this.dragHelper.active = false;
               break;
    }
}
Game.prototype.handleMouseDown = function(e) {
    var x, y;
    e.preventDefault();

    if (Modernizr.touch) {
        x = e.touches[0].pageX;
        y = e.touches[0].pageY;
    } else {
        x = e.clientX;
        y = e.clientY;
    }

    switch (Tools.current) {
          case Tools.BUILD:

               break;
          case Tools.MOVE:
               this.dragHelper.active = true;
               this.dragHelper.x = x;
               this.dragHelper.y = y;
               break;
          case Tools.ZOOM_IN:
               this.zoomIn();
               break;
          case Tools.ZOOM_OUT:
               this.zoomOut();
               break;
          case Tools.DEMOLISH:
```

```
                var pos = this.translatePixelsToMatrix(x, y);

                if (this.tileMap[pos.row] != undefined &&
                  this.tileMap[pos.row][pos.col] != undefined) {
                      this.tileMap[pos.row][pos.col] = null;
                }

                break;
    }
    this.draw();
}

Game.prototype.doResize = function() {

    this.canvas.width = document.body.clientWidth;
    this.canvas.height = document.body.clientHeight;


    this.draw();
}

Game.prototype.translatePixelsToMatrix = function(x, y) {
    var tileHeight = this.tile.height * this.zoomHelper.level;
    var tileWidth = this.tile.width * this.zoomHelper.level;
    var zoomedHeight = (this.grid.height * this.zoomHelper.level);
    var gridOffsetY = zoomedHeight + this.scrollPosition.y;
    var gridOffsetX = (this.grid.width * this.zoomHelper.level);

    //默认情况下，网格水平居中
    var zoomedWidth = ((tileWidth / 2) * this.zoomHelper.level;
    gridOffsetX += (this.canvas.width / 2) - zoomedWidth) + this.
       scrollPosition.x;

    var col = (2 * (y - gridOffsetY) - x + gridOffsetX) / 2;
    var row = x + col - gridOffsetX - tileHeight;

    col = Math.round(col / tileHeight);
    row = Math.round(row / tileHeight);

    return {
          row: row,
          col: col
    }
}

Game.prototype.draw = function(srcX, srcY, destX, destY) {
    srcX = (srcX === undefined) ? 0 : srcX;
    srcY = (srcY === undefined) ? 0 : srcY;
    destX = (destX === undefined) ? this.canvas.width : destX;
    destY = (destY === undefined) ? this.canvas.height : destY;

    this.c.clearRect (0, 0, this.canvas.width, this.canvas.height);
    this.c.fillStyle = '#0C3B00'; // 绿色背景
    this.c.fillRect (0, 0, this.canvas.width, this.canvas.height);
```

```
var pos_TL = this.translatePixelsToMatrix(1, 1);
   var pos_BL = this.translatePixelsToMatrix(1, this.canvas.height);
   var pos_TR = this.translatePixelsToMatrix(this.canvas.width, 1);
   var pos_BR = this.translatePixelsToMatrix(this.canvas.width,
                                              this.canvas.height);

   var startRow = pos_TL.row;
   var startCol = pos_TR.col;
   var rowCount = pos_BR.row + 1;
   var colCount = pos_BL.col + 1;

   startRow = (startRow < 0) ? 0 : startRow;
   startCol = (startCol < 0) ? 0 : startCol;

   rowCount = (rowCount > this.grid.width) ? this.grid.width : rowCount;
   colCount = (colCount > this.grid.height) ? this.grid.height : colCount;

   var tileHeight = this.tile.height * this.zoomHelper.level;
   var tileWidth = this.tile.width * this.zoomHelper.level;

   for (var row = startRow; row < rowCount; row++) {
         for (var col = startCol; col < colCount; col++) {
               var xpos = (row - col) * tileHeight + (this.grid.width *
             this.zoomHelper.level);
               xpos += (this.canvas.width / 2) - ((tileWidth / 2) *
             this.zoomHelper.level) + this.scrollPosition.x;

               var ypos = (row + col) * (tileHeight / 2) +
                  (this.grid.height *
             this.zoomHelper.level) + this.scrollPosition.y;

               if (this.tileMap[row] != null && this.tileMap[row]
                  [col] != null) {
                     // 放置建筑物
               } else {
                     if (Math.round(xpos) + tileWidth >= srcX &&
                         Math.round(ypos) + tileHeight >= srcY &&
                         Math.round(xpos) <= destX &&
                         Math.round(ypos) <= destY) {


                         this.c.drawImage(this.tile,
                                       Math.round(xpos),
                                       Math.round(ypos),
                                       tileWidth,
                                       tileHeight);

                     }
               }
         }
}
```

```
}

Game.prototype.zoomIn = function() {
    switch(this.zoomHelper.level) {
          case this.zoomHelper.NORMAL:
               this.zoomHelper.level = this.zoomHelper.CLOSE;
               break;
          case this.zoomHelper.FAR:
               this.zoomHelper.level = this.zoomHelper.NORMAL;
               break;
          case this.zoomHelper.CLOSE:
               return;
    }

    //居中视图
    this.scrollPosition.y -= (this.grid.height * this.zoomHelper.level) +
       this.scrollPosition.y;
    this.scrollPosition.x -= (this.grid.width * this.zoomHelper.level) +
        this.scrollPosition.x;
}

Game.prototype.zoomOut = function() {
    switch(this.zoomHelper.level) {
          case this.zoomHelper.NORMAL:
               this.zoomHelper.level = this.zoomHelper.FAR;
               break;
          case this.zoomHelper.CLOSE:
               this.zoomHelper.level = this.zoomHelper.NORMAL;
               break;
          case this.zoomHelper.FAR:
               return;
    }

    //居中视图
    this.scrollPosition.y -= (this.grid.height * this.zoomHelper.level) +
        this.scrollPosition.y;
    this.scrollPosition.x -= (this.grid.width * this.zoomHelper.level) +
        this.scrollPosition.x;
}

Game.prototype.rotateGrid = function(mW, mH, sW, sH) {
    var m = [];


    mW = (mW === undefined) ? this.grid.width : mW;
    mH = (mH === undefined) ? this.grid.height : mH;

    sW = (sW === undefined) ? 0 : sW;
    sH = (sH === undefined) ? 0 : sH;

    for (var i = sW; i < mW; i++) {
        for (var j = sH; j < mH; j++) {
```

```
            var row = (mW - j) - 1;

            if (this.tileMap[row] !== undefined && this.tileMap[row][i]) {
                 m[i] = (m[i] === undefined) ? [] : m[i];
                 m[i][j] = this.tileMap[row][i];
            }
        }
    }

    this.tileMap = m;
}
```

第 4 章

HTML5声音及处理优化



HTML5 为开发人员带来的不仅仅是 canvas 元素。原生支持声音（以及视频）也很重要，这样在编写游戏时，我们就能像处理图形一样，利用相同的 JavaScript 环境来管理声音。其他对 JavaScript 的增强还包括是否通过 Web Worker 把一个大任务分成几个小任务来执行，以及通过本地及会话存储机制在用户的设备上保存信息。

4.1 通过audio元素添加声音

HTML 之前的规范支持几种在页面中添加音频文件的方式。

- 使用 object 或 embed 标签在页面中嵌入文件或插件——例如 LiveAudio（Netscape Navigator）或 ActiveMovie Control（Internet Explorer）。随着时间推移，其他插件进入了人们的视野，比如 Macromedia Flash（现在的 Adobe Flash）、REAL Player 或苹果公司的 QuickTime，等等。要在网页中嵌入一段 MIDI 或 WAV 音频，可以使用 <embed src="music.mid" autostart="true" loop="true">，或者嵌入一个第三方插件，比如 Macromedia Flash Player 通过 SWF 文件来播放声音。
- 插入一个 Java Applet 并通过它来播放声音。
- 向页面的 body 元素添加 bgsound 属性（仅 Internet Explorer 支持）。

总之，通常就是在浏览器中包含一些插件，而这就意味着我们的声音在有的浏览器中可以播放，但在其他浏览器中可能就无法播放——即使这些浏览器都安装在同一台计算机中也不行。

好在 HTML5 来了，它为我们带来了通过 <audio> 和 <video> 标签原生播放音频和视频文件的能力。

然而，HTML 之前版本的一些限制也被 HTML5 的一些限制所取代。音频（还有视频）是利用不同的编解码器进行编码和解码的。编解码器是一个小型的软件库，可以对实现了特定算法的音频或视频的数据文件或流进行编码和解码。有的算法目的在于优化速度，而有的算法是为了保证品质无损。但正如常见的软件一样，其中有些算法不用交版税即可使用，而另一些算法则必须取得许可才能使用。

对于“开放”编解码器，苹果和微软等软件公司担心会侵犯专利权，导致吃官司——这也是他们为什么不支持某些编解码器的原因。而另外一些公司（例如 Mozilla Foundation 或 Opera Software）则恰好相反，尚未就某些使用许可达成必要的协议。

图 4-1 展示了各种浏览器支持的音频编解码器。

	MP3	WAV	OGG
Chrome	✓	✗	✓
Explorer 9	✓	✓	✗
Firefox	✗	✓	✓
Opera	✗	✓	✓
Safari	✓	✓	✗

图 4-1　浏览器对音频编解码器的支持

但愿等到 HTML5 规范制定完成时，所有浏览器供应商都能够达成协议，使某种通用的编解码器能够在所有平台上工作。在此期间，W3C 提供了一种处理当前问题的优雅的方式，即允许定义“后备的音频来源”。

要在网页中嵌入一个音频播放器，只需创建一个 audio 标签即可：

```
<audio src="../sounds/song.ogg" type="audio/ogg" controls />
```

这行代码会显示一个带播放 / 暂停控件（由标签中的 controls 属性指定）的音频播放器。按下“播放”按钮，播放器会尝试播放 src 属性指定的声音。但是，有可能我们使用的浏览器不支持那种文件格式 / 编解码器，此时可以这样做：

```
<audio controls>
  <source src="../sounds/song.mp3" type="audio/mpeg">
  <source src="../sounds/song.ogg" type="audio/ogg">
</audio>
```

区别在于没有使用唯一的 src 属性，因为 HTML5 的 audio 标签允许定义多个音频文件。如果出于某种原因 song.mp3 不能播放，浏览器就会尝试播放后备的 son.ogg。在定义了多个音频文件的情况下，浏览器会尝试播放其中的每一个文件，直到全部失败或者其中的某一个能够播放。除了 controls 属性，HTML5 的 audio 标签还支持另外几个可选的属性：

`loop`

指定循环播放 src 属性或单独的源标签中指定的媒体文件

`autoplay`

指定在加载完媒体文件之后立即播放

`preload`

指定预先加载媒体文件的方式

`preload="none"`

不预先加载文件，而是用户单击播放按钮时加载

`preload="metadata"`

仅预先加载文件的元数据

`preload="auto"`

由浏览器决定是否预先加载文件（通常意味着预先加载整个文件）

当然，也可以使用 JavaScript 来创建并使用 HTML5 的 `audio` 对象，而不是在文档中包含 `audio` 元素，如程序示例 4-1 所示。

程序示例 4-1 使用 JavaScript 创建 HTML5 的 `audio` 对象

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>Example 15 (HTML5 Audio)</title>
    <script>
      window.onload = function() {

        // 定义一个音频文件的数组，供浏览器尝试
        var sources = [
          ["../sounds/song.mp3", "audio/mpeg"],
          ["../sounds/song.ogg", "audio/ogg"]
        ];

        // 创建 HTML5 的 audio 元素
        var audio = document.createElement('audio');

        // 循环 sources 数组
        for (var i = 0; i < sources.length; i++) {

          // 稍后将介绍如何检测浏览器是否支持某种格式
          // 创建一个 source 参数
          var src = document.createElement('source');
          // 添加 src 和 type 属性
          src.setAttribute("src", sources[i][0]);
          src.setAttribute("type", sources[i][1]);
          // 将 source 元素添加到 audio 元素中
          audio.appendChild(src);
        }
```

```
            // 尝试播放声音
            audio.play();

          }
        </script>
      </head>
      <body>
        HTML5 Audio tag example.
      </body>
    </html>
```



本书在线代码库中还包含一个更有效的例子，文件名是 ex15-html5Audio-alt.html。这个例子采用了另一种方法，检测浏览器是否支持相应的音频格式。

对于 HTML5 的 audio 和 video 对象，主流浏览器都会触发一组新的事件，叫做媒体事件（media event），要了解相关的所有事件请参考 http://t.cn/akH7dT[1]。

以下是本书及我们的游戏中将会用到的一些事件：

canplaythrough

在文件下载几乎完成、能够完整播放的时候触发

playing

告诉我们声音是否正在播放

ended

在播放完成时触发

HTML5 音频（及视频）文件的播放可以通过下列方法和属性来控制：

play()

播放媒体

pause()

暂停播放媒体

currentTime

用于取得或设置当前的播放时间，以毫秒为单位

注 1：长 URL：https://developer.mozilla.org/En/Using_audio_and_video_in_Firefox。（译者注）

`volume`

用于取得或设置当前的音量，以 0.0 ～ 1.0 之间的值表示



Mozilla Foundation 有一个正在进行中的项目名叫 Audio Data API，这个 API 可以让开发人员创建和操作声音时获得更好的准确性和更细微的控制。（在本书写作时，这个 API 只有 Firefox 和最新版本的 Chromium 支持。）更多信息请参考：https://wiki.mozilla.org/Audio_Data_API。

知道了 HTML5 对声音的支持情况及其相关的控制选项之后，下面我们了解一下它的限制：

- 使用 HTML5 的 `audio` 对象创建和播放声音时，只能播放一次。如果你想让同样的声音同步播放两次，就需要另外创建一个 `audio` 对象；
- 同时播放的文件数量也有限制；这个限制在不同的平台中也不一样。如果超过这个限制，不同的平台也会给出不同的错误。

根据经验，同时播放的音频数量最多不要超过 3 个（或更少），因为这是移动操作系统对播放数量的限制。其他平台（如 PC 机上的 Firefox）支持同时播放的音频数量更多一些。

在讨论 HTML5 Canvas 的部分，我们曾把几幅图像组合成一张图片（称为精灵表），用于减少到服务器的请求数量。下载完精灵表之后，就可以通过由（*X1*, *Y1*）及（*X2*, *Y2*）[这里的（*X1*, *Y1*）及（*X2*, *Y2*）是像素坐标] 界定的矩形从中取得特定的图像。对于声音文件，我们也可以采取类似的方式把它们组合成一个文件（称为*声音表*）。只不过在取得其中特定的声音文件时，使用的不是像素坐标，而是时间坐标。（假如你的想象力够丰富，也可以在左右音频声道中播放不同的声音，这样甚至可以进一步减少请求，但代价就是放弃双声道而仅用单声道播放声音。）

在我们的游戏中，将使用一个名为 SoundUtil 的实用程序库来处理声音表，让它负责操作一个音频对象池，以提高内存及资源使用效率。

SoundUtil 库使用的方法与例 7 中的方法不同——没有创建 `audio` 标签，而是创建 `audio` 对象。在请求这个程序库播放声音时，需要传递下列参数：

- 一个包含文件本身以及每个文件编码格式的数组
- 开始时间，以秒为单位
- 结束时间，以毫秒为单位
- 音量值

- 一个表示是否希望声音不断循环播放的布尔值，如果传入 true 则只关注第一次播放的开始时间，不关注结束时间

这个库的 play() 方法将会调用另一个方法 getAudioObject()，后者会操作一个可供重用的音频对象池，并跟踪能够同时播放音频的最大的数量值。如果池中没有可用对象则自动生成一个，除非池中对象的数量等于能够同时播放的音频的最大值——在这种情况下，该方法会返回一个 null 值，不会播放任何声音。

当声音播放完之后，需要调用 freeAudioObject() 方法“释放”音频对象，即将音频对象放到可用音频对象池中。

程序示例 4-2 展示了这个完整的实用程序库。

程序示例 4-2　SoundUtil.js

```
// 音频对象池中可以容纳的最大对象数
var MAX_PLAYBACKS = 6;
var globalVolume = 0.6;

function SoundUtil(maxPlaybacks) {
  this.maxPlaybacks = maxPlaybacks;
  this.audioObjects = [];// 可供重复利用的音频对象池
}

SoundUtil.prototype.play = function(file, startTime, duration, volume,
   loop) {

  // 从池中取得音频对象
  var audioObject = this.getAudioObject();
  var suObj = this;

  /**
   * 池中没有可用的音频对象。无法播放
   * 注意：这只是平常情形采用的手段
   * 稍后你也可以向这个队列中添加对象
   */
  if (audioObject !== null) {
    audioObject.obj.loop = loop;
    audioObject.obj.volume = volume;

    for (var i = 0; i < file.length; i++) {
      if (audioObject.obj.canPlayType(file[i][1]) === "maybe" ||
        audioObject.obj.canPlayType(file[i][1]) === "probably") {
        audioObject.obj.src = file[i][0];
        audioObject.obj.type = file[i][1];
        break;
      }
    }

    var playBack = function() {
      // 删除事件监听器，否则它就会不断被调用
```

```
      audioObject.obj.removeEventListener('canplaythrough', playBack, false);
      audioObject.obj.currentTime = startTime;
      audioObject.obj.play();

      // 在循环模式下，如果对象播放完成，则无须监听
      if (!loop) {
        setTimeout(function() {
          audioObject.obj.pause();
          suObj.freeAudioObject(audioObject);
        }, duration);
      }
    }

    audioObject.obj.addEventListener('canplaythrough', playBack, false);
  }
}

SoundUtil.prototype.getAudioObject = function() {
  if (this.audioObjects.length === 0) {
    var a = new Audio();
    var audioObject = {
      id: 0,
      obj: a,
      busy: true
    }

    this.audioObjects.push (audioObject);

    return audioObject;
  } else {
    for (var i = 0; i < this.audioObjects.length; i++) {
      if (!this.audioObjects[i].busy) {
        this.audioObjects[i].busy = true;
        return this.audioObjects[i];
      }
    }

    // 没有释放音频对象。能否创建一个新的音频对象？
    if (this.audioObjects.length <= this.maxPlaybacks) {
      var a = new Audio();
      var audioObject = {
        id: this.audioObjects.length,
        obj: a,
        busy: true
      }

      this.audioObjects.push (audioObject);

      return audioObject;
    } else {
        return null;
    }
  }
}
```

```
SoundUtil.prototype.freeAudioObject = function(audioObject) {
    for (var i = 0; i < this.audioObjects.length; i++) {
        if (this.audioObjects[i].id === audioObject.id) {
            this.audioObjects[i].currentTime = 0;
            this.audioObjects[i].busy = false;
        }
    }
}
```

为了演示如何使用 SoundUtil 库，我们将把它用在本书第一个例子“标题界面”（程序示例 1-1）中。程序示例 4-3 展示的是本书在线代码库 examples 文件夹中 ex16-soundUtil.html 文件的代码。

程序示例 4-3　向标题界面中添加声音

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>Example 16 - Title Screen with Sound</title>

    <!-- We're included the soundutil as an external file -->
    <script src="soundutil.js" charset="utf-8"></script>
    <script>

      window.onload = function () {
        var su = null;
        var sources = [
          ["../sounds/title.mp3", "audio/mp3"],
          ["../sounds/title.ogg", "audio/ogg"]
        ];

        var canvas = document.getElementById('myCanvas');
        var c = canvas.getContext('2d');

        var State = {
          _current: 0,
          INTRO: 0,
          LOADING: 1,
          LOADED: 2
        }

        window.addEventListener('click', handleClick, false);
        window.addEventListener('resize', doResize, false);

        doResize();
        // 检测当前浏览器是支持 MP3，还是支持 OGG
        if (soundIsSupported()) {
          // 播放标题界面音乐
          playTitleMusic();
        }

        function playTitleMusic() {
          if (su) {
```

```
    su.play(sources, 0, 156000, globalVolume, false);
  }
}

function soundIsSupported() {
  var a = new Audio();
  var failures = 0;

  for (var i = 0; i < sources.length; i++) {
    if (a.canPlayType(sources[i][1]) !== "maybe" &&
      a.canPlayType(sources[i][1]) !== "probably") {
      failures++;
    }
  }

  if (failures !== sources.length) {
    su = new SoundUtil()
    return true;
  } else {
    return false;
  }
}

function handleClick() {
  if (State._current !== State.LOADING) {
    State._current = State.LOADING;
    fadeToWhite();
  }
}

function doResize() {
  canvas.width = document.body.clientWidth;
  canvas.height = document.body.clientHeight;

  switch (State._current) {
    case State.INTRO:
      showIntro ();
      break;
  }
}

function fadeToWhite(alphaVal) {
  // 如果函数没有接收到任何参数，从 0.02 开始
  var alphaVal = (alphaVal == undefined) ? 0.02 : parseFloat(alphaVal)
    + 0.02;
  // 将颜色设置为白色
  c.fillStyle = '#FFFFFF';
  // 设置 globalAlpha 属性
  c.globalAlpha = alphaVal;

  // 绘制一个跟画布一样大的矩形
  c.fillRect(0, 0, canvas.width, canvas.height);

  if (alphaVal < 1.0) {
```

```
        setTimeout(function() {
          fadeToWhite(alphaVal);
        }, 30);
      } else {
        State._current = State.LOADED;
      }
    }

    function showIntro () {
      var phrase = "Click or tap the screen to start the game";

      // 清除画布
      c.clearRect (0, 0, canvas.width, canvas.height);

      // 创建一个好看的蓝色渐变
      var grd = c.createLinearGradient(0, canvas.height, canvas.width, 0);
      grd.addColorStop(0, '#ceefff');
      grd.addColorStop(1, '#52bcff');

      c.fillStyle = grd;
      c.fillRect(0, 0, canvas.width, canvas.height);

      var logoImg = new Image();
      logoImg.src = '../img/logo.png';

      // 保存原始的宽度值，以便将来使用相同的宽高比
      var originalWidth = logoImg.width;

      // 计算新的宽度和高度值
      logoImg.width = Math.round((50 * document.body.clientWidth) / 100);
      logoImg.height = Math.round((logoImg.width * logoImg.height) /
        originalWidth);
      // 创建一个小辅助对象
      var logo = {
        img: logoImg,
          x: (canvas.width/2) - (logoImg.width/2),
          y: (canvas.height/2) - (logoImg.height/2)
        }

        // 展示图像
        c.drawImage(logo.img, logo.x, logo.y, logo.img.width, logo.img.
            height);
        // 把颜色修改为黑色
        c.fillStyle = '#000000';
        c.font = 'bold 16px Arial, sans-serif';

        var textSize = c.measureText (phrase);
        var xCoord = (canvas.width / 2) - (textSize.width / 2);

        c.fillText (phrase, xCoord, (logo.y + logo.img.height) + 50);
      }
    }
  </script>
  <style type="text/css" media="screen">
    html { height: 100%; overflow: hidden }
    body {
      margin: 0px;
```

```
        padding: 0px;
        height: 100%;
      }
    </style>

  </head>
  <body>
    <canvas id="myCanvas" width="100" height="100">
      Your browser doesn't include support for the canvas tag.
    </canvas>
  </body>
</html>
```

我们会修改 ex14-gui.html，以便为游戏添加背景音乐。修改后的例子请参见在线代码库中的 ex14-gui-sound.html。

4.2 用Web Workers API执行大计算量任务

我们一直致力于寻求一种高性能的图形渲染方法，以便将其用于最终的游戏。而路径查找则是一个非常有用的功能，可以用于创建道路或显示角色从 A 点到 B 点的过程。

简言之，路径查找算法就是要在 *n* 维（通常是 2D 或 3D）空间中找出两点间的最短路线。

通常，只有少数人才能实现准确的路径查找；换句话说，很多人（但愿不包括我们）都无法保证计算结果的准确性。可以说这是一项计算量很大的工作，而最有效的解决方案就是根据我们的产品修改算法，以便得到最合用的方法。

处理路径查找的一种最佳算法叫做 A*，是迪杰斯特拉（Dijkstra）算法的变体。路径查找（或者类似的计算时间超过数毫秒的操作）的问题在于，它们会导致 JavaScript 产生一种名为“界面锁定”的效果，也就是在操作完成以前，浏览器将一直被冻结。

幸运的是，HTML5 规范也提供了一个名为 Web Workers 的新 API。Web Workers（通常称为“worker”）可以让我们在后台执行计算量相对较大以及执行时间较长的脚本，而不会影响浏览器的主用户界面。



worker 不是银弹，不能魔幻般地让原来吃掉 100% 的 CPU 处理能力的任务变得轻而易举。只要是常规手段下计算量大的任务，使用 worker 可能照样还是计算量大，最终还是会影响到用户体验。不过，如果是只消耗了 30% 的 CPU 处理能力的任务，利用 worker 并行来处理还是可以把用户界面的影响降到最低的。

当然，也有一些限制：

- 由于每个 worker 都运行在与执行它们的页面完全独立、线程安全的环境下(也叫“沙盒”)，因此它们不能访问 DOM 和 `window` 对象；
- 尽管已有的 worker 可以再产生新的 worker（谷歌的 Chrome 不支持这个功能），但使用起来必须多加小心，因为这样一来，如果产生 bug 将会很难调试。

要创建 worker，可以使用以下语法：

```
var worker = new Worker(PATH_TO_A_JS_SCRIPT);
```

其中的 *`PATH_TO_A_JS_SCRIPT`* 可以是一个脚本文件，比如 astar.js。在创建了 worker 之后，任何时候调用 `worker.close()` 都可以终止它的执行。如果终止了一个 worker，然后又需要执行一个新操作，那么就得再创建一个新的 worker 对象。

Web Workers 之间的通信是通过在 `worker.onmessage` 事件的回调函数中调用 `worker.postMessage(object)` 来实现的。此外，还可以通过 `onerror` 事件处理程序来处理 worker 的错误。

与普通的网页类似，Web Workers 也支持引入外部脚本，使用的是 `importScripts()` 函数。这个函数接受零个或多个参数，如果有参数，每个参数都应该是一个 JavaScript 文件。

本书在线代码库的 ex17-grid-astar.html 中有一个用 JavaScript 实现的 A^* 算法，其中使用了 Web Worders。图 4-2 展示了这个示例的运行效果。程序示例 4-4 和程序示例 4-5 展示了网页及 A^* 算法的 JavaScript 实现。

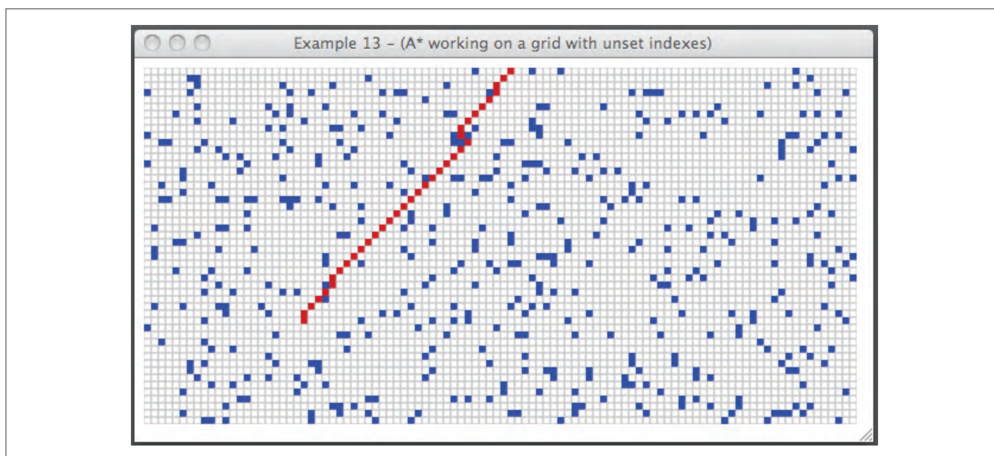


图 4-2　程序示例 4-4 的运行效果

程序示例 4-4　路径查找 HTML

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>Example 17 - (A* working on a grid with unset indexes using
           web workers)</title>

    <script>

      window.onload = function () {
        var tileMap = [];

        var path = {
          start: null,
          stop: null
        }

        var tile = {
          width: 6,
          height: 6
        }

        var grid = {
          width: 100,
          height: 100
        }

        var canvas = document.getElementById('myCanvas');
        canvas.addEventListener('click', handleClick, false);
        var c = canvas.getContext('2d');

        // 随机生成 1000 个元素
        for (var i = 0; i < 1000; i++) {
          generateRandomElement();
        }

        // 绘制整个网格
        draw();

        function handleClick(e) {
          // 检测到鼠标单击后，把鼠标坐标转换为像素坐标
          var row = Math.floor((e.clientX - 10) / tile.width);
          var column = Math.floor((e.clientY - 10) / tile.height);

          if (tileMap[row] == null) {
            tileMap[row] = [];
          }

          if (tileMap[row][column] !== 0 && tileMap[row][column] !== 1) {
            tileMap[row][column] = 0;

            if (path.start === null) {
              path.start = {x: row, y: column};
```

```
        } else {
          path.stop = {x: row, y: column};

          callWorker(path, processWorkerResults);

          path.start = null;
          path.stop = null;
        }

        draw();
      }
    }

    function callWorker(path, callback) {
      var w = new Worker('astar.js');
      w.postMessage({
        tileMap: tileMap,
        grid: {
          width: grid.width,
          height: grid.height
        },
        start: path.start,
        stop: path.stop
      });
      w.onmessage = callback;
    }

    function processWorkerResults(e) {
      if (e.data.length > 0) {
      for (var i = 0, len = e.data.length; i < len; i++) {
        if (tileMap[e.data[i].x] === undefined) {
          tileMap[e.data[i].x] = [];
        }

        tileMap[e.data[i].x][e.data[i].y] = 0;
      }
    }

    draw();
  }

  function generateRandomElement() {
    var rndRow = Math.floor(Math.random() * (grid.width + 1));
    var rndCol = Math.floor(Math.random() * (grid.height + 1));


    if (tileMap[rndRow] == null) {
      tileMap[rndRow] = [];
    }
    tileMap[rndRow][rndCol] = 1;
  }

  function draw(srcX, srcY, destX, destY) {
    srcX = (srcX === undefined) ? 0 : srcX;
    srcY = (srcY === undefined) ? 0 : srcY;
    destX = (destX === undefined) ? canvas.width : destX;
```

```
        destY = (destY === undefined) ? canvas.height : destY;

        c.fillStyle = '#FFFFFF';
        c.fillRect (srcX, srcY, destX + 1, destY + 1);
        c.fillStyle = '#000000';

        var startRow = 0;
        var startCol = 0;
        var rowCount = startRow + Math.floor(canvas.width / tile.
           width) + 1;
        var colCount = startCol + Math.floor(canvas.height / tile.
           height) + 1;

        rowCount = ((startRow + rowCount) > grid.width) ? grid.width :
           rowCount;
        colCount = ((startCol + colCount) > grid.height) ? grid.height :
           colCount;

        for (var row = startRow; row < rowCount; row++) {
          for (var col = startCol; col < colCount; col++) {
            var tilePositionX = tile.width * row;
            var tilePositionY = tile.height * col;

            if (tilePositionX >= srcX && tilePositionY >= srcY &&
              tilePositionX <= (srcX + destX) &&
              tilePositionY <= (srcY + destY)) {

              if (tileMap[row] != null && tileMap[row][col] != null) {
                if (tileMap[row][col] == 0) {
                c.fillStyle = '#CC0000';
              } else {
                c.fillStyle = '#0000FF';
                }

                c.fillRect(tilePositionX, tilePositionY, tile.width,
                    tile.height);
              } else {
                c.strokeStyle = '#CCCCCC';
                c.strokeRect(tilePositionX, tilePositionY, tile.width,
                    tile.height);
              }
            }
          }
        }
      }
    }
  </script>
  </head>
  <body>
  <canvas id="myCanvas" width="600" height="300"></canvas>
  <br />

  </body>
</html>
```

程序示例 4-5　A* 算法的 JavaScript 实现

```
// 这个 worker 处理负责 aStar 类的实例
onmessage = function(e){
  var a = new aStar(e.data.tileMap, e.data.grid.width, e.data.grid.height,
    e.data.start, e.data.stop);
  postMessage(a);
}


// 基于非连续索引的 tileMap 调整后的 A* 路径查找类

/**
 * @param tileMap: A 2-dimensional matrix with noncontiguous indexes
 * @param gridW: Grid width measured in rows
 * @param gridH: Grid height measured in columns
 * @param src: Source point, an object containing X and Y
 *             coordinates representing row/column
 * @param dest: Destination point, an object containing
 *              X and Y coordinates representing row/column
 * @param createPositions: [OPTIONAL] A boolean indicating whether
 *                         traversing through the tileMap should
 *                         create new indexes (default TRUE)
 */
var aStar = function(tileMap, gridW, gridH, src, dest, createPositions) {
  this.openList = new NodeList(true, 'F');
  this.closedList = new NodeList();
  this.path = new NodeList();
  this.src = src;
  this.dest = dest;
  this.createPositions = (createPositions === undefined) ? true :
       createPositions;
  this.currentNode = null;

  var grid = {
    rows: gridW,
    cols: gridH
  }

  this.openList.add(new Node(null, this.src));

  while (!this.openList.isEmpty()) {
    this.currentNode = this.openList.get(0);
    this.currentNode.visited = true;

    if (this.checkDifference(this.currentNode, this.dest)) {
      // 到达目的地 :)
      break;
    }

    this.closedList.add(this.currentNode);
    this.openList.remove(0);

    // 检查与当前节点相近的 8 个元素
    var nstart = {
```

```
    x: (((this.currentNode.x - 1) >= 0) ? this.currentNode.x - 1 : 0),
    y: (((this.currentNode.y - 1) >= 0) ? this.currentNode.y - 1 : 0),
  }

  var nstop = {
    x: (((this.currentNode.x + 1) <= grid.rows) ? this.currentNode.
        x + 1 : grid.rows),
    y: (((this.currentNode.y + 1) <= grid.cols) ? this.currentNode.
        y + 1 : grid.cols),
  }

  for (var row = nstart.x; row <= nstop.x; row++) {
    for (var col = nstart.y; col <= nstop.y; col++) {

    // 在原始的 tileMap 中还没有行，还继续吗？
    if (tileMap[row] === undefined) {
      if (!this.createPositions) {
        continue;
      }
    }

    // 检查建筑物或其他障碍物
    if (tileMap[row] !== undefined && tileMap[row][col] === 1) {
      continue;
    }

    var element = this.closedList.getByXY(row, col);
    if (element !== null) {
      // 这个元素已经在 closedList 中了
      continue;
    } else {
      element = this.openList.getByXY(row, col);
      if (element !== null) {
        // 这个元素已经在 closedList 中了
      continue;
      }
    }
      // 还不在任何列表中，继续
      var n = new Node(this.currentNode, {x: row, y: col});
      n.G = this.currentNode.G + 1;
      n.H = this.getDistance(this.currentNode, n);
      n.F = n.G + n.H;

      this.openList.add(n);
    }
  }
}

while (this.currentNode.parentNode !== null) {
  this.path.add(this.currentNode);
  this.currentNode = this.currentNode.parentNode;
}

return this.path.list;
```

```
}

aStar.prototype.checkDifference = function(src, dest) {
  return (src.x === dest.x && src.y === dest.y);
}

aStar.prototype.getDistance = function(src, dest) {
  return Math.abs(src.x - dest.x) + Math.abs(src.y - dest.y);
}

function Node(parentNode, src) {
  this.parentNode = parentNode;
    this.x = src.x;
    this.y = src.y;
    this.F = 0;
    this.G = 0;
    this.H = 0;
}

var NodeList = function(sorted, sortParam) {
  this.sort = (sorted === undefined) ? false : sorted;
  this.sortParam = (sortParam === undefined) ? 'F' : sortParam;
  this.list = [];
  this.coordMatrix = [];
}

NodeList.prototype.add = function(element) {
  this.list.push(element);

  if (this.coordMatrix[element.x] === undefined) {
    this.coordMatrix[element.x] = [];
  }

  this.coordMatrix[element.x][element.y] = element;

  if (this.sort) {
    var sortBy = this.sortParam;
    this.list.sort(function(o1, o2) { return o1[sortBy] - o2[sortBy]; });
  }
}

NodeList.prototype.remove = function(pos) {
  this.list.splice(pos, 1);
}

NodeList.prototype.get = function(pos) {
  return this.list[pos];
}

NodeList.prototype.size = function() {
  return this.list.length;
}

NodeList.prototype.isEmpty = function() {
  return (this.list.length == 0);
```

```
}

NodeList.prototype.getByXY = function(x, y) {
  if (this.coordMatrix[x] === undefined) {
    return null;
  } else {
    var obj = this.coordMatrix[x][y];

    if (obj == undefined) {
      return null;
    } else {
      return obj;
    }
  }
}
NodeList.prototype.print = function() {
  for (var i = 0, len = this.list.length; i < len; i++) {
    console.log(this.list[i].x + ' ' + this.list[i].y);
  }
}
```

4.3 本地存储和会话存储

过去，开发人员经常会碰到的一个问题就是使用 cookie 不能保存太多信息（最多只有 4 K），而且保存的信息也不可能很重要。而今天，主流浏览器开始支持 Web Storage 了，利用它可以在用户的本地硬盘上保存至少 5 MB 的数据。虽然说“至少 5 MB”，但根据浏览器不同，这个数量实际上可能会多一些，也可能会少一些。在某些浏览器（比如 Opera）中，这个空间配额是可以在设置面板中指定的。在某些情况下，Web Storage 也可能会被用户完全禁用。假如存储的数据量超过了 5 MB，那浏览器就会抛出 `QUOTA_EXCEEDED_ERR` 的异常。为此，一定别忘了把 `localStorage` 或 `sessionStorage` 封装在一个 try-catch 块中，就像其他处理异常的代码一样。

可想而知，Web Storage 与 cookie 很大程度上异曲同工：

- 二者都可能会被禁用；
- 存储的数据量都有最大限制——cookie 是 4 K，而 Web Storage 是 5 MB 或者更少一些；
- 用户随时可以删除或手工创建 / 修改存储目录中的内容；
- 浏览器也可能会自动让存储目录中的内容“过期”；
- 空间配额是以域名为单位分配的，而且由所有子域共享（也就是 site1.example.com、site2.example.com 和 site3.example.com 共享同一个目录）。

不过，Web Storage 与 cookie 也存在如下差别：

- 通过 Web Storage API 保存的数据只能在客户端查询，不能被服务器端查询；
- 存储的内容不会随着每个请求“来来回回”传递；
- 除了 sessionStroage（它在会话结束时删除自己保存的数据）以外，不能明确指定数据的过期时间。

还有一点与使用 cookie 时一样，即最好不要使用 Web Storage 来保存重要的内容。

除此之外，Web Storage 也确实能够帮我们做到之前做不到的一些事儿。例如，缓存对象以提升用户下次打开应用时的加载速度、保存正在撰写的文件草稿，甚至还可以把它当成虚拟的内存容器来使用。

在明白了 Web Storage 的优点和不足之后，剩下的问题也就简单了：

- 它以键值对数组的形式保存数据，所有数据都以字符串形式存储；
- localStorage 与 sessionStorage 之间的区别是前者永久保存数据（或保存到用户或浏览器处理它们），而后者只在“会话”期间保存数据（关闭标签页或窗口后就没有了）。

Web Storage 的 API 只有以下 4 个方法：

`localStorage.setItem(key, value)`

添加数据项

`localStorage.getItem(key)`

查询已有的数据项

`localStorage.removeItem(key)`

删除特定的数据项

`localStorage.clear()`

完全删除 localStorage 目录中的内容

程序示例 4-6 展示了一个使用 Web Storage 的例子。

程序示例 4-6　使用 Web Storage

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>Canvas Example 18 (LocalStorage)</title>

    <script>
```

```
window.onload = function () {
  var binMatrix = null;
  var matrixSize = 25;

  // 这个键名用于保存、加载和删除数据
  var KEY_NAME = "matrix";

  var matrix = document.getElementById('matrix');
  var load = document.getElementById('load');
  var save = document.getElementById('save');
  var clear = document.getElementById('clear');

  binMatrix = initializeMatrix(matrixSize);
  printMatrix(binMatrix, matrix);

  // 处理单击按钮的事件
  load.addEventListener('click', handleLoad, false);
  save.addEventListener('click', handleSave, false);
  clear.addEventListener('click', handleClear, false);

  function handleLoad() {
    var m = localStorage.getItem(KEY_NAME);

    try {
      // 如果没有相应的键，或者已经删除了变量 m 的内容，就会返回 null
      if (m == null) {
        alert("You haven't stored a matrix yet.");
      } else {
        // 否则，需要把内容“解析”回数组形式
        binMatrix = JSON.parse(m);

        // 清除原始的矩阵
        matrix.innerHTML = null;

        // 重新输出
        printMatrix(binMatrix, matrix);
      }
    } catch(e) {
      alert("The following error occurred while trying to load
           the matrix: " + e);
    }
  }

  function handleSave() {
    try {
        // 读取“矩阵”<div> 中复选框的值并循环记录在数组中
      for (var i = 0; i < matrixSize; i++) {
        for (var j = 0; j < matrixSize; j++) {
          var pos = (i + j) + (i * matrixSize);

           if (matrix.childNodes[pos].tagName == "INPUT") {
            binMatrix[i][j] = (matrix.childNodes[pos].checked)
                ? 1 : 0;
```

```
            }
          }
        }
        // 最后，把数据变成字符串以便保存
        localStorage.setItem(KEY_NAME, JSON.stringify(binMatrix));
      } catch(e) {
        alert("The following error occurred while trying to save
          the matrix: " + e);
      }
    }

    function handleClear() {
      if (confirm("Are you sure that you want to empty the
        matrix?")) {
        try {
          localStorage.removeItem(KEY_NAME);

          // 清除原始的矩阵
          matrix.innerHTML = null;
          binMatrix = null;

          // 重新生成矩阵
          binMatrix = initializeMatrix(matrixSize);

          // 重新输出
          printMatrix(binMatrix, matrix);
        } catch(e) {
          alert("The following error occurred while trying to remove
            the matrix: " + e);
      }
    }
  }
}

/**
 * 矩阵的初始化函数
 */
function initializeMatrix(size) {
  var m = [];

  for (var i = 0; i < size; i++) {
    m[i] = [];
  for (var j = 0; j < size; j++) {
    m[i][j] = 0;
  }
}

  return m;
}
/**
 * 这个函数取得矩阵并将其转换成长长的复选框字符串，然后将其插入到“矩阵”<div>中。
 * 这是一种最佳实践，除非真的有采取其他方法的必要，否则，使用字符串来生成HTML元素
 * 可以省去不少创建新元素的麻烦。
 * 把所有字符串拼接到一块，再“一次性地”把它插入到文档中，可以避免不必要的同时也是
 * 严重影响浏览器性能的页面重绘。
 */
```

```
    function printMatrix(m, elem) {
      var str = "";

      for (var i = 0, x = m.length; i < x; i++) {
        for (var j = 0, r = m[i].length; j < r; j++) {
          str += '<input type="checkbox" class="' + i + ' - ' + j + '" ';
          str += (m[i][j] == 1) ? 'checked' : '';
          str += ' />';
          str += ((j + 1) == r) ? '<div class="clb"></div>' : '';
        }
      }

      elem.innerHTML = str;
    }
  </script>

  <style type="text/css" media="screen">
    body {
      margin: 20px;
      padding: 0px;
    }

    #matrix input {
      float: left;
      padding: 0px;
      margin: 0px;
    }
    div.clb { clear: both; }
  </style>
</head>
<body>
  <input type="button" id="load" value="Load Matrix" />
  <input type="button" id="save" value="Save Matrix" />
  <input type="button" id="clear" value="Clear Matrix" />

  <br /><br />
  <div id="matrix"></div>
</body>
</html>
```

程序示例4-6的完整代码也保存在本地在线代码库example文件夹下的ex18-localStorage.html文件中。



要了解有关Web Storage的更多信息，请参考该规范的在线页面（在本书写作时仍然是一个草案）：http://dev.w3.org/html5/webstorage/。

我们不打算在本书的游戏中使用`localStorage`。如果你在自己开发游戏时需要绘制包含大量元素的网格，还是建议你“一部分一部分地”处理包含所有物体的矩阵。只要修改一下书中的代码，就可以实现随着网格滚动从服务器上下载额外的区块，然后将它们保存到`localStorage`中，随时准备补充到当前的矩阵中去。

第 5 章

推向市场



你已经使用交互的图形和音乐构建了一个有吸引力的游戏。现在要做的就是准备一番，邀请人们来玩了！换句话说，你得把游戏放到服务器上，防止有人（按照他们的意愿）作弊修改代码，然后把游戏连接到一个玩家众多的地方——Facebook！

5.1 预防作弊及服务器端工作

开发在线视频游戏需要考虑的一个主要问题就是预防作弊。这就跟一般的 Web 开发类似，我们不能相信任何用户，因此，保证应用不受恶意用户破坏，处理不期而至的输入或返回值，永远都是第一要务。

对于源代码开放的游戏，特别是使用 JavaScript 和 HTML 等 Web 技术构建的游戏而言，这种风险无疑又加大了。因为要修改变量（甚至修改 POST/GET 请求）非常容易，动态实时地修改客户端的代码也并非难事。

更令人不安的是，针对这个问题的解决方案又因游戏不同而不同。不过，任何方案都要依赖于两个重要（同时效率也通常非常低）的手段，必须在游戏开发之前以及开发过程中加以运用：

- 把在客户端提交恶意数据的风险降到最低；
- 在服务器端验证一切。

至于我们的游戏，以及大多数实时社交策略游戏，都需要考虑如下问题：

- 每个用户的账户余额都要保存在数据库的字段或者表格中（取决于你是否想要跟踪每一桩交易），而每一次购买和卖出操作都要相应地更新该余额；
- 有必要在服务器上保存一份 Unix 时间戳，并定期与客户端进行同步；
- 最重要的一件事（以及反作弊的最可靠方法），也许就是把游戏设计得能够在服务器上可靠地预测用户在任意时刻的分值，而无须与客户端交互。

怎么才能做到这些呢？我们来看一看下面的场景。

最初的时候，用户的账户余额是 2000 个金币、0 座建筑物、账户“创建时间”为 1293861660（这是表示 2011 年 1 月 1 日 0 时 0 分 0 秒的 Unix 时间戳）、“最后更新”时间为 1293861660（与开始时间相同）。

用户可以建造 3 种建筑物：

- 花 250 个金币建一个冰激凌店，每 30 分钟付 5 个金币；
- 花 1000 个金币建一个酒店，每小时付 30 个金币；
- 花 500 个金币建一个电影院，每 30 分钟付 12 个金币。

用户在 1293861660（2011 年 1 月 1 日 0 时 1 分 0 秒，即创建账户后 1 分钟）建了一个冰激凌店。

后来，他在 1294084800（2011 年 1 月 3 日 14 时 0 分 0 秒）又建了一个酒店。

在 1294120800（2011 年 1 月 4 日 0 时 0 分 0 秒），这个用户又建一个电影院。

在 1294639200（2011 年 1 月 10 日 0 时 0 分 0 秒），他登录自己的账号，想查一查账户余额。

对于上述情况，可以这样处理：跟踪用户购买的所有建筑物，在用户账户中添加“最后更新”字段，以便知道用户查询账户余额、购买或卖出的最后时间；然后，在他每次查询账户余额或执行购买 / 卖出操作时，都重复如下步骤：

(1) 用当前的时间戳更新用户的“最后更新”字段；
(2) 遍历用户拥有的全部建筑物；
(3) 计算当前时间与用户“最后更新”时间之间的差值（以秒计）；
(4) 用这个差除以单位付款时段，然后将结果向下舍入；
(5) 用得到的结果乘以单位付款时段需要向用户支付的金币数；
(6) 更新账户余额，加上计算结果；
(7) 根据操作不同（显示、建造或卖出），或者显示账户余额，或者减去 / 加上建筑物的购置费；
(8) 如果是卖出建筑物，则删除建筑物与账户的关联。

把这些步骤应用到前面的情景，可以作出如下推断。

- 时间为 1293861660 的时候，用户要建一个冰激凌店。此时，他拥有 0 个建筑物，因此只要更新他的账户余额、“最后更新”时间并创建关联即可。本次关联的“最后更新”时间为 1293861660。

  当前用户账户余额是 1750（2000−250）。

- 时间为 1294084800 的时候，用户想建一个酒店。我们要为这个酒店创建一个新的关联、重新计算他的账户余额。我们知道他在 1293861660 的时候购买了一个冰激凌店，现在把“最后更新”字段更新为 1294084800；为了得出冰激凌店到目前为止产生了多少收入，要进行如下计算：

  ```
  timeDiff = 1294084800 - 1293861660 = 223140 秒
  ```

  冰激凌店每 30 分钟（1800 秒）产生 5 个金币的收益，因此要进一步执行如下计算：

  ```
  result = timeDiff / 1800 秒 = 123.97
  ```

也可以向下舍入成 123 并计算总收益：

```
result = result * 5 = 615 个金币
```

最后将用户余额更新为 1750 + 615 = 2365、创建与酒店的关联（以 1294084800 作为“最后更新”时间）并收取酒店的购置费。用户账户的当前余额为 1365（2365−1000）。

- 在时间为 1294120800 的时候，用户购买了一个电影院，因此要重复与刚才相同的步骤（这一次要计算两个建筑物——冰激凌店和酒店）：

```
timeDiff = 1294120800 - 1294084800 = 36000 秒
iceCreamShopResult = timeDiff / 1800 = 20
iceCreamShopResult = iceCreamShopResult * 5 个金币 = 100 个金币
hotelResult = timeDiff / 3600 秒（1 小时）= 10
hotelResult = hotelResult * 30 金币 = 300 个金币
accBalance = accBalance + hotelResult + iceCreamShopResult = 1765
accBalance = accBalance - 500 个金币（电影院的购置费）
```

用户账户此时的余额为 1265。

- 最后，在时间为 1294639200 的时候，用户想再查询一下自己的账户余额：

```
timeDiff = 1294639200 - 1294120800 = 518400 秒
iceCreamShopResult = timeDiff / 1800 = 288
iceCreamShopResult = iceCreamShopResult * 5 = 1440 个金币
hotelResult = timeDiff / 3600秒（1 小时）= 144
hotelResult = hotelResult * 30 = 4320 个金币
cinemaResult = timeDiff / 1800 = 288
cinemaResult = cinemaResult * 12 = 3456 个金币
accBalance = accBalance + cinemaResult + hotelResult + iceCreamShopResult
```

也就是说，到 2011 年 1 月 10 日 0 时 0 分 0 秒，我们可以确定用户的账户余额是 10481。

虽然想要阻止用户修改游戏的客户端极其困难，但也并非绝对不可能。像我们刚才分析的这样，就可以在某种程度上防止恶意用户谋取不正当的优势。因为他们的修改只能影响自己本地，不会对服务器造成影响。另外，还可以采用 Zynga 曾在他们的游戏中使用过的另一种有用的功能，即不像前面讲的那样随时计算收入，而是让用户自己去“收集”建筑物产生的收益。举个例子，假设某建筑物每 30 分钟生成 500 个金币，而用户 3 天没有玩游戏，那么在他收集金币时，只能得到 500 个金币的奖励。为此，我们只要比较根据时间差计算的金币数是否比建筑物在单位付款时段内生成的金币数多即可；如果是，则将一个标志设置为 `true`。之后，用户就必须手工“收集”金币，将该标记重置为 `false`。

我们可以用图 5-1 展示的数据模型来实现之前展示的最初的方法。

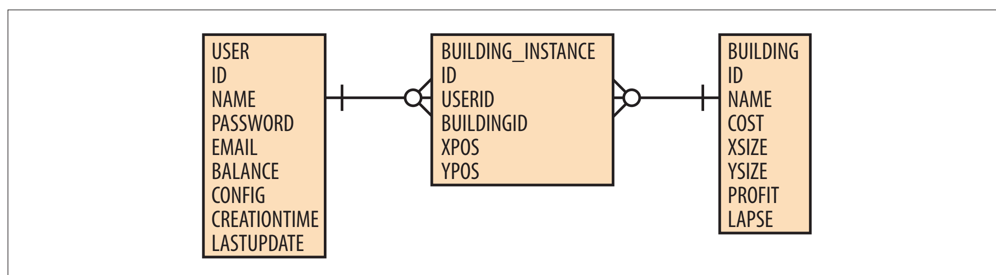


图 5-1　关联用户与建筑物的数据模型

图 5-1 所示的数据模型告诉我们：

任何用户都可以与零个或多个建筑物实例建立关系；
任何个人都必须有一个用户账号；
任何建筑物也都可以与零个或多个建筑物实例建立关系；
任何建筑物实例都必须与一个建筑物建立关系。

在我们的游戏中，以上数据模型可以使用 MySQL 实现，逻辑控制可以使用 PHP 来实现[1]：

- 关于在本地安装和配置 MySQL 的说明，请参见 http://dev.mysql.com/usingmysql/get_started.html；
- 关于在本地安装和配置 PHP 的说明，请参见 http://www.php.net/manual/zh/install.php；为了运行 PHP，还需要再安装一个 Web 服务器，如 Apache、Lighttpd 或 nginx，关于如何安装（和配置）这些 Web 服务器的说明也可以在前面以 .php 结尾的 URL 中找到。

在你的计算机中安装并配置好 PHP 及 MySQL（包括设置了 root 密码）之后，打开新的命令行终端并通过如下命令连接到服务器：

```
mysql -hlocalhost -uroot -p<密码>
```

如果没有为 root 用户设置密码，可以试试这个命令：

```
mysql -hlocalhost -uroot
```

连接到服务器之后，就可以使用 MySQL 的命令提示符了：

注 1：建议读者安装 XAMPP，“XAMPP 是一个易于安装且包含 MySQL、PHP 和 Perl 的 Apache 发行版”。下载地址为：http://www.apachefriends.org/zh_cn/xampp.html。（译者注）

```
scarlet:~ andres$ mysql -uroot -hlocalhost
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 816
Server version: 5.1.45 MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the current input
   statement.

mysql>
```



你安装的 MySQL 服务器的版本可能会有所不同。

连接到数据库之后，先使用如下命令来为游戏创建一个数据库：

```
CREATE DATABASE mygame;
```

如果一切顺利，应该看到如下输出结果：

```
mysql> CREATE DATABASE mygame;
Query OK, 1 row affected (0.00 sec)
```

为了在这个数据库中创建表，首先必须告诉 MySQL 选择该数据库作为当前数据库：

```
USE mygame;
```

如果没有发生什么错误，就会看到如下返回值：

```
mysql> USE mygame;
Database changed
```

在本书在线代码库的 server 目录中（https://github.com/andrespagella/Making-Isometric-Real-time-Games/tree/master/server），可以找到两个 .sql 文件：model-empty.sql 和 model-filled.sql。把 model-filled.sql 下载到你的计算机中，然后返回到 MySQL 命令提示符下。用下载文件的路径替换以下语句中的占位符文本并执行该命令：

```
source <下载文件的路径>;
```

如果一切顺利的话，应该可以看到类似下列的输出：

```
mysql> source /Users/andres/Desktop/model-filled.sql
Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

Query OK, 4 rows affected (0.05 sec)
```

```
Query OK, 0 rows affected (0.05 sec)

Query OK, 0 rows affected (0.07 sec)

Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.10 sec)

Query OK, 0 rows affected (0.09 sec)

mysql>
```

这个 SQL 文件会生成 3 个数据库表：users、buildings 和 building_instances。同时，还会向 buildings 表中加入 4 种建筑物（将在最终的游戏中使用）：冰激凌店、酒店、电影院和树。

为了在 PHP 脚本中操作 mygame 数据库，还需要创建一个 MySQL 用户，并为该用户赋予执行 SELECT、INSERT、UPDATE 和 DELETE 的权限。在 MySQL 命令提示符下执行如下命令：

```
CREATE USER 'mygameuser'@'localhost' IDENTIFIED BY 'game123';
GRANT SELECT, INSERT, UPDATE, DELETE ON mygame.* TO
   'mygameuser'@'localhost';
```

跟以前一样，输出中也不应该包含什么错误：

```
mysql> CREATE USER 'mygameuser'@'localhost' IDENTIFIED BY 'game123';
Query OK, 0 rows affected (0.09 sec)

mysql> GRANT SELECT, INSERT, UPDATE, DELETE ON mygame.* TO
   'mygameuser'@'localhost';
Query OK, 0 rows affected (0.07 sec)

mysql>
```



关于如何使用 MySQL 的更多信息，请参考 MySQL 官方开发人员门户网站：http://dev.mysql.com。

在 GitHub 代码库的 server 目录中，还可以看到如下文件和目录：

config.php

用户定义数据库连接的细节信息以及网格的大小

classes/class.dbutil.php

简单的 MySQL 数据库实用工具类

classes/class.users.php

处理用户相关的操作

classes/class.buildings.php

处理建筑物相关的操作

classes/class.operations.php

负责实例化和检索建筑物的实例

classes/class.user.php

用户（`User`）类

classes/class.building.php

建筑物（`Building`）类

classes/class.buildingInstance.php

建筑物实例（`BuildingInstance`）类

test-database.php

测试数据库连接以及数据库是否可以存储、检索和删除记录的脚本

registration.php

基于前面介绍的类来实现用户注册的脚本

authentication.php

基于前面介绍的类来实现用户认证及发起用户会话的脚本

5.2 通往最终游戏的路

上一节，我们学习了怎么实现游戏的服务器端脚本及数据库结构。本节将把目前开发好的游戏与服务器端脚本结合，处理用户注册、认证，并使用数据库中真实存在的数据来填充“建筑物”面板或账户余额中的值。

游戏最终的文件和目录结构如下。

index.php

包含游戏首页，处理认证和用户注册。

game.php

包含实际的游戏代码。如果会话结束，则把用户带回首页面去认证和注册。这个文件中的代码与程序示例 3-1 中的代码很相似 ，只不过为了在几个容器中显示真实的数据而作了些修改，同时也为了方便维护而把 JavaScript 拆成了几个文件。

async/

其中的 PHP 脚本用于处理 JavaScript 通过 XMLHttpRequest（AJAX）发出的异步调用。

css/

包含 CSS 文件 site.css（用于 index.php）和 ui-style.css（用在游戏中）。

js/

包含游戏要用到的所有 JavaScript 文件。

除了为所有字段指定真实的数据之外，game.php 还会多做一些工作，取得与当前用户关联的所有建筑物实例（`BuildingInstance`），并以用户购买的这些建筑物来填充 `tileMap` 矩阵。此外，出于装饰的目的，index.php 中的代码也会随机地创建“树”（Tree）建筑物实例（网格中 10% 的区域将“种”上这些树）。

这些建筑物实例在网格中的初始化工作由函数 `initializedGrid()` 负责，该函数将包含建筑物实例的 PHP 数组与游戏中的 `tileMap` 矩阵组合到一起。除非用户打算用建筑物来填满整个网格（这意味着他必须要买下 62 500 个建筑物，因此可能性极小），否则这种方法已经足够满足我们的要求了。或者也可以在 PHP 中创建一个完全相同的 `tileMap` 矩阵，编码成 JSON，然后（在 JavaScript 中）解码 JOSN 并替换原始的 `tileMap` 矩阵。不过，根据我的测试，这种方法的效率很低，因为在 JavaScript 中解码大型的 JSON 对象开销非常大。



假如你想在自己的或者专业的项目中使用更大的网格，最好是把初始化过程拆开，先只加载整个网格的一部分，然后再基于画布滚动动态加载其余部分。我们在这里之所以没有实现这种方法，主要是因为移动设备往往下载带宽不错，但往返服务器（或者说连接服务器）的延迟很明显（尤其是通过 3G 连接的时候）。换句话说，在这种情况下最好一次性下载所有资源，而不是经常通过“小型的”请求去获取更多数据。

我们还打算对 Game 类稍作修改，以防它一完成初始化就显示网格。修改涉及删除 Game() 构造函数中的两行代码：

```
this.doResize()
this.draw()
```

此时此刻，另一个需要完善的地方是修改标题界面的示例，使其在用户加载完 game.php 页面后立即显示。要添加标题界面，就意味着必须再增加一个对象来跟踪当前游戏状态（LOADING、LOADED、PLAYING），而这正是我们想要创建 GameState 对象（全局可用）的原因，它的定义如下：

```
var GameState = {
  _current: null,
  LOADING: 0,
  LOADED: 1,
  TITLESCREEN: 2,
  PLAYING: 3
}
```

然后，根据 GameState._current 中保存的当前游戏状态，某些对象和事件的行为会有所不同。比如，在 GameState._current 中的状态是 GameState.LOADING 的情况下，用户不能触发任何事件。而在 GameState._current 被设置为 GameState.TITLESCREEN 时，单击页面就会显示游戏。

一开始，标题界面会充当一个缓冲器，在后台预先加载图像、声音等游戏资源。因此，我们想要使用一个名为 ResourceLoader 的对象，通过它在使用资源之前先下载和预加载所有文件。

以下是 ResourceLoader 类的定义，这个在 resourceLoader.js 文件中定义的类非常直观：

```
//ResourceLoader 类的定义

var ResourceType = {
  IMAGE: 0,
  SOUND: 1
}

function ResourceLoader(onPartial, onComplete) {
  this.resources = [];
  this.resourcesLoaded = 0;

  if (onPartial !== undefined && typeof(onPartial) === "function") {
    this.onPartial = onPartial;
  }

  if (onComplete !== undefined && typeof(onComplete) === "function") {
```

```
    this.onComplete = onComplete;
  }
}

ResourceLoader.prototype.addResource = function(filePath, fileType,
   resourceType) {
  var res = {
    filePath: filePath,
    fileType: fileType,
    resourceType: resourceType
  };

  this.resources.push(res);
}

ResourceLoader.prototype.startPreloading = function() {
  for (var i = 0, len = this.resources.length; i < len; i++) {
    switch(this.resources[i].resourceType) {
      case ResourceType.IMAGE:
        var img = new Image();
        var rl = this;

        img.src = this.resources[i].filePath;
        img.addEventListener('load', function() { rl.onResourceLoaded(); },
           false);
        break;
    case ResourceType.SOUND:
        var a = new Audio();

        // 只预先加载需要播放的声音文件
        if (a.canPlayType(this.resources[i].fileType) === "maybe" ||
          a.canPlayType(this.resources[i].fileType) === "probably") {

          a.src = this.resources[i].filePath;
          a.type = this.resources[i].fileType;

          var rl = this;
          a.addEventListener('canplaythrough', function() {
            a.removeEventListener('canplaythrough', arguments.callee, false);
            rl.onResourceLoaded();
          }, false);
        } else {
          // 无法播放声音。认为资源加载完毕。
          this.onResourceLoaded();
        }

        break;
    }
  }
}

ResourceLoader.prototype.onResourceLoaded = function() {
  this.resourcesLoaded++;

  if (this.onPartial != undefined) {
```

```
      this.onPartial();
    }

    if (this.resourcesLoaded == this.resources.length) {
      if (this.onComplete != undefined) {
        this.onComplete();
      }
    }

    return;
  }

  ResourceLoader.prototype.isLoadComplete = function() {
    if (this.resources.length == this.resourcesLoaded) {
      return true;
    }

    return false;
  }
```

可以像下面这样使用 `ResourceLoader` 类：

```
var rl = new ResourceLoader();

rl.addResource('image1.png', null, ResourceType.IMAGE);
rl.addResource('image2.jpg', null, ResourceType.IMAGE);
rl.addResource('image3.gif', null, ResourceType.IMAGE);
rl.addResource('sound.ogg', 'audio/ogg', ResourceType.SOUND);
rl.addResource('sound.mp3', 'audio/mp3', ResourceType.SOUND);

rl.startPreloading();
```

只要调用了 `ResourceLoader` 类的 `startPreloading()` 方法，马上就可以访问两个非常有用的属性：

`ResourceLoaderInstance.resources.length`

返回添加后的资源数量

`ResourceLoaderInstance.resourcesLoaded`

返回迄今为止已经预加载的资源数量

把这两个变量的值结合起来，就可以计算出任何时刻已经预加载完成的资源百分比，计算方法如下：

```
var percentLoaded = Math.floor((ResourceLoaderInstance.resourcesLoaded * 100) /
  ResourceLoadedInstance.resources.length);
```

除此之外，`ResourceLoader` 类还支持两个回调方法：

`onPartial`

每加载完一个元素时调用

onComplete

所有资源都加载完毕时调用

使用这两个回调方法，可以在标题界面显示加载进度条，如图 5-2 所示。

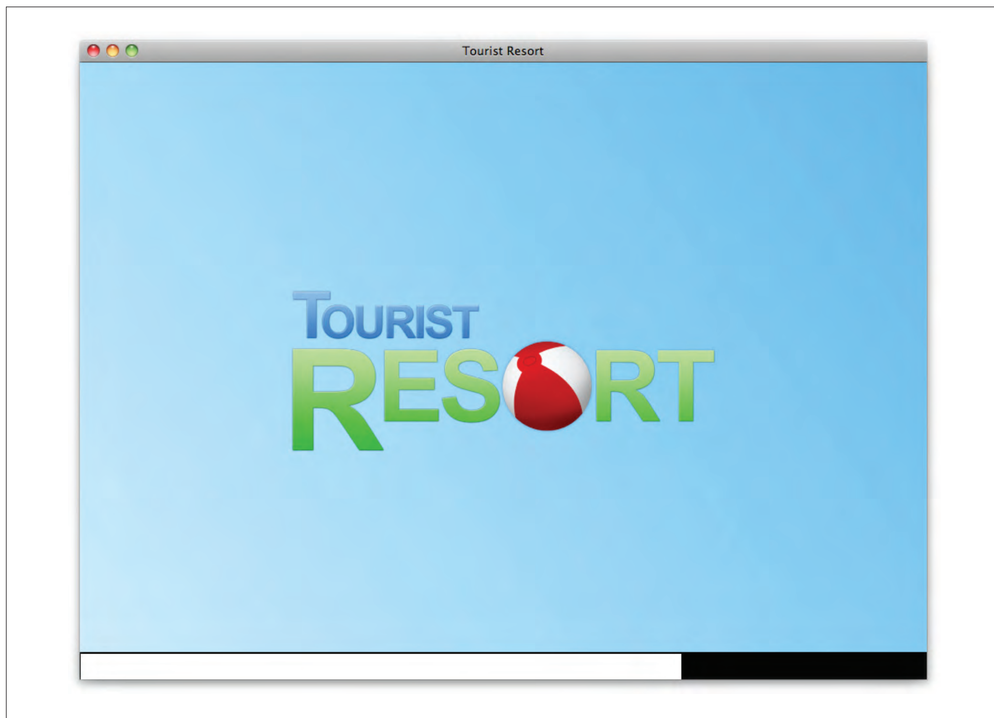


图 5-2　带有加载进度条的标题界面

游戏会自行管理不同的状态，即根据 GameState 保存的当前状态执行不同的代码。为了实现游戏状态之间的转换，我们使用一个名为 handleGameState() 的函数，让它在文档加载完毕后执行。这个函数使用之前展示的 GameState 对象，其定义如下：

```
function handleGameState(nextState) {
  if (nextState !== undefined) {
    GameState._current = nextState;
  }

  switch(GameState._current) {
    case GameState.LOADING:
      // ...
      break;
```

```
    case GameState.LOADED:
      // ...
      break;
    case GameState.TITLESCREEN:
      // ...
      break;
    case GameState.PLAYING:
      // ...
      break;
    default:
      // ...
      break;
  }
}
```

如果想改变当前游戏的状态，直接指定一个有效的 `GameState` 值即可。比如，可以直接添加一个表示暂停页面的 `GameState.PAUSED` 状态，当用户按了 Esc 键时会转换为这个状态。然后，用户再按一次 Esc 键就调用 `handleGameState(GameState.PLAYING)`，返回游戏。

如果是在 `GameState.PLAYING` 状态下，则调用 `preloadResources()` 函数来实例化一个 `ResourceLoader` 对象，让它来负责预先加载运行游戏所需的各种资源（图片和声音文件）。这个函数的代码如下：

```
function preloadResources(canvas, callback) {
  var c = canvas.getContext('2d');

  var rl = new ResourceLoader(printProgressBar, callback);

  rl.addResource('../img/tile.png', null, ResourceType.IMAGE);
  rl.addResource('../img/ui-icons.png', null, ResourceType.IMAGE);
  rl.addResource('../img/spritesheet.png', null, ResourceType.IMAGE);

  rl.addResource('../sounds/title.ogg', 'audio/ogg', ResourceType.SOUND);
  rl.addResource('../sounds/title.mp3', 'audio/mp3', ResourceType.SOUND);

  rl.startPreloading();

  printProgressBar();

  function printProgressBar() {
    var percent = Math.floor((rl.resourcesLoaded * 100) / rl.resources.
      length);

    var cwidth = Math.floor((percent * (canvas.width - 1)) / 100);

    c.fillStyle = '#000000';
    c.fillRect(0, canvas.height - 30, canvas.width, canvas.height);

    c.fillStyle = '#FFFFFF';
    c.fillRect(1, canvas.height - 28, cwidth, canvas.height - 6);
  }
}
```

这个函数会在页面中输出如图 5-2 所示的进度条。

可能有读者注意到了，这里并没有加载 cinema.png or tree.png，加载的是 spritesheet.png。为了减少请求的数量（减少请求数量是优化 Web 应用性能的一个主要任务），我们没有单独地加载每一张图片，而是把这些图片都组合到了一张图片中，这张大图片会由本书前面展示的 `Sprint` 对象使用。

创建资源加载器（`ResourceLoader`）的实例时也传入了一个名为 `callback` 的参数，该参数无非就是一个 `handleGameState(GameState.LOADED)` 调用，以便加载完资源后转换到下一个状态。

`GameState.LOADED` 状态下的代码将负责创建“Game”实例，同时用当前用户所有的建筑物实例来填充区块地图矩阵。在加载完成矩阵之后，它仍然会自动调用 `handleGameState()`，传入 `GameState.TITLESCREEN` 以便转换到下一个状态。

`GameState.TITLESCREEN` 状态表示显示游戏的标题界面（在我们的游戏中，标题界面会显示“Tourist Resort”标志和一句话“click or tap the screen to start the game”）。与此同时，该状态下的代码会为“单击事件”添加一个监听程序，用户单击一次屏幕就会将其删除。用户单击 / 触摸窗口的任何部位后，游戏会显示一系列淡入淡出动画，在白色背景上显示一句话：“Developed by you.”最后，动画一结束，就会再次调用 `handleGameState()` 并传入 `GameState.PLAYING`。于是就显示游戏的 UI 及带有全部建筑物的网格，然后监听事件。

在请求 game.php（包含游戏）文件时，服务器会自动填充建筑物面板，以显示可供建造的建筑物、费用、收入、时间等。图 5-3 展示了这个面板的屏幕截图。

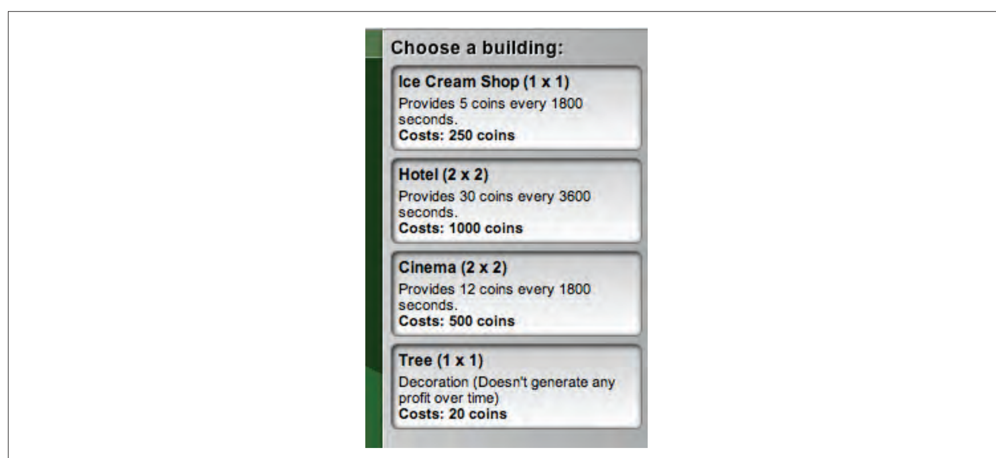


图 5-3　建筑物面板为用户提供的选项

好了，现在我们开始监听事件了。那么像滚动、鼠标抬起 / 按下、按键按下等输入事件都需要监听，为了管理单击事件发生时触发什么操作，还要使用一个类似 `GameState` 的对象，用于跟踪用户当前选择的工具：

```
var Tools = {
    current: 4, // 默认工具
    MOVE: 0,
    ZOOM_IN: 1,
    ZOOM_OUT: 2,
    DEMOLISH: 3,
    SELECT: 4,
    BUILD: 5
}
```

然后，在用户单击网格时，就会触发下面这个事件处理程序，进而恰当地处理用户的动作：

```
Game.prototype.handleMouseDown = function(e) {
  e.preventDefault();

  switch (Tools.current) {
    case Tools.BUILD:
      // ...
      break;
    // 其他工具，略……
  }
}
```

某些动作，比如建造和拆掉，也需要在服务器上更新相应的数据库字段。为了与服务器通信，需要用到下面这个 comm.js 文件中的函数：

```
var SERVER_PATH_URL = "http://localhost:8080/";

function request(url, callback) {
  var req = false;

  if (window.XMLHttpRequest) {
    try {
        req = new XMLHttpRequest();
    } catch(e) {
        // 什么也不做
    }
  }

  if (req) {
    req.open("GET", url, true);
    req.send(null);
    req.onreadystatechange = function() {
      switch(req.readyState) {
        case 2:
          if (req.status !== 200) {
```

```
            callback('ERROR');
            return;
          }
          break;
        case 4:
          callback (req.responseText);
          break;
        }
      }
    } else {
      // 不支持 XMLHttpRequest
      callback('ERROR');
    }
  }

  // 购买
  function purchase(buildingId, row, col, callback) {
    var url = SERVER_PATH_URL + 'purchase.php';

    url += "?buildingId=" + buildingId;
    url += "&x=" + row;
    url += "&y=" + col;

    request(url, callback);
  }

  // 拆掉
  function demolish(row, col) {
    var url = SERVER_PATH_URL + 'demolish.php';

    url += "?x=" + row;
    url += "&y=" + col;

    request(url, callback);
  }

  // 同步
  function sync() {
    var url = SERVER_PATH_URL + 'sync.php';

    request(url, callback);
  }
```

使用这些函数非常简单。要购买一棵树（`buildingId=4`）放在第 4 行第 5 列，可以使用如下代码：

```
purchase(4, 4, 5, function(resp) {
  if (resp.substr(0, 3) == 'OK:') {
    var buildingInstanceId = resp.substr(3, resp.length);
   alert("Building purchase was successful. The instance ID is " +
      buildingInstanceId);
  } else {
   alert("An error occurred while trying to purchase the building!")
  }
});
```

不过，在购买建筑物之前，还得检查被单击的区块或它周围的其他区块是否已经被另一个建筑物（或 BuildPortion）占用了。为此，需要用到 Game 对象的 Game.checkIfTileIsBusy() 方法：

```
Game.prototype.checkIfTileIsBusy = function(obj, row, col) {
  for (var i = (row + 1) - obj.tileWidth; i <= row; i++) {
    for (var j = (col + 1) - obj.tileHeight; j <= col; j++) {
      if (this.tileMap[i] != undefined && this.tileMap[i][j] != null) {
        return true;
      }
    }
  }

  return false;
}
```

既然我们现在已经有了能够恰当处理各种事件的工具，那么接下来所要做的就是完成函数 Game.prototype.handleMouseDown 的最终实现：

```
Game.prototype.handleMouseDown = function(e) {
  e.preventDefault();

  switch (Tools.current) {
    case Tools.BUILD:
      if (this.buildHelper.current != null) {
        var pos = this.translatePixelsToMatrix(e.clientX, e.clientY);

        // 可以在这个网格上面放建筑物吗?
        if (!this.checkIfTileIsBusy(this.buildHelper.current, pos.row,
          pos.col)) {

          var obj = this.buildHelper.current;
          var t = this;

          var processResponse = function(resp) {
            if (resp.substr(0, 3) == 'OK:') {
              var buildingInstanceId = resp.substr(3, resp.length);
              for (var i = (pos.row + 1) - obj.tileWidth; i <= pos.row;
                i++) {
                for (var j = (pos.col + 1) - obj.tileHeight; j <= pos.
                  col; j++) {
                  t.tileMap[i] = (t.tileMap[i] == undefined) ? [] :
                      t.tileMap[i];

                  t.tileMap[i][j] = (i === pos.row && j === pos.col) ? obj
:
                      new BuildingPortion(obj.buildingTypeId, i, j);
                }
              }
            } else {
              alert("An error occurred while trying to purchase the
                building!")
            }

            t.draw();
          }
```

```
        purchase(obj.buildingTypeId, pos.row, pos.col, processResponse);
        } else {
          alert("Unable purchase building on this position");
        }
      }

      break;
    case Tools.MOVE:
      this.dragHelper.active = true;
      this.dragHelper.x = e.clientX;
      this.dragHelper.y = e.clientY;
      break;
    case Tools.ZOOM_IN:
      this.zoomIn();
      break;
    case Tools.ZOOM_OUT:
      this.zoomOut();
      break;
    case Tools.DEMOLISH:

      var pos = this.translatePixelsToMatrix(e.clientX, e.clientY);

      if (this.tileMap[pos.row] != undefined && this.tileMap[pos.row]
         [pos.col] !=undefined) {
        var obj = this.tileMap[pos.row][pos.col];

        // 不是整个建筑物，只是建筑物的一部分。取得对原始建筑物的引用
        if (obj instanceof BuildingPortion) {
          pos.row += obj.x;
          pos.col += obj.y;
          obj = this.tileMap[pos.row][pos.col];
        }

        var t = this;
        var processResponse = function(resp) {
          if (resp.substr(0, 2) == 'OK') {
            // 检查周围的像素，同时毁掉 BuildingPortion
            for (var i = (pos.row + 1) - obj.tileWidth; i <= pos.row;
              i++) {
              for (var j = (pos.col + 1) - obj.tileHeight; j <= pos.col;
                j++) {
                t.tileMap[i][j] = null;
              }
            }
          } else {
            alert("A problem occurred while trying to demolish this
              building");
          }

          t.draw();
        }

        demolish(pos.row, pos.col, processResponse);
      }

      break;
  }

  this.draw();
}
```

虽然现在可以购买或者拆掉建筑物（根据用户操作在数据库中添加或删除建筑物实例），但账户余额的值始终都是相同的。为了确保更新这个值，还需要实现一个名为 refresh() 的函数，每 15 秒钟调用它一次，以便更新账户余额。在服务器端，sync.php 也会负责计算由建筑物生成的金币数：

```
function refresh() {
  var processResponse = function(resp) {
    if (resp.substr(0, 5) == 'ERROR') {
      alert("A problem occurred while trying to sync with the service.");
    } else {
      var balanceContainer = document.getElementById('balance');

      var currBalance = parseInt(balanceContainer.innerHTML);
      var balance = parseInt(resp.substr(3, resp.length));
      balanceContainer.innerHTML = balance;
    }
  }

  sync(processResponse);
  setTimeout(function() {
    refresh();
  }, 15000);
}
```

图 5-4 展示了此时游戏的屏幕截图。



图 5-4　绿荫环抱的度假胜地

5.3 对游戏作最后修饰

所谓最后的修饰，指的是让产品（对于我们而言，就是游戏）显得与众不同。虽然目前的游戏已经具备了相应的功能，但仍然会让人觉得有点简陋、无聊，甚至没心思去玩它。

有一个办法可以让游戏看起来更有活力，那就是在城市上空添加一些漂浮的物体，比如云彩。我们这次不用 `canvas` 来显示它们（否则就得在云彩漂移过程中不断重绘画布上的一切），而使用 CSS3 动画来实现这一效果。

利用 CSS3 动画，可以像下面这样修改一些关键帧中的一或多个 CSS 属性：

```
@moveToRight {
  0% {
    left: 0px
  }

  50% {
    left: 100px
  }

  100% {
    left: 200px
  }
}
```

定义了动画关键帧（在此，相应的元素将从位置 `left:0px` 开始，移动到位置 `left:200px`），还需要定义其他动画属性：

`animation-timing-function`

控制关键帧变换为下一帧的方式。可能的值为 `ease`、`linear`、`ease-in`、`ease-out`、`ease-in-out`、`cubic-bezier(`*number*, *number*, *number*, *number*`)`。

`animation-name`

指定动画的名字（在此，我们起名为 `moveToRight`）。

`animation-duration`

控制整个动画持续的时间。

`animation-iteration-count`

接受一个整数或“infinite”，控制动画播放的次数。

`animation-direction`

用于定义动画的“方向”。可能的值包括 `normal`（从关键帧 0 到关键帧 100）或 `alternate`（从关键帧 0 到关键帧 100，再从关键帧 100 到关键帧 0）。

全部属性都可以在 W3C 的工作草案页面查到：http://www.w3.org/TR/css3-animations/。

虽然在我们这个游戏中，动画只在几个固定的位置上发生，不过使用 JavaScript 和随机变量或者基于游戏中当前网格滚动位置的变量，可以轻易创造出相同的效果来：

```
@-webkit-keyframes moveFromLeftToRight {
  0% {
    -webkit-transform: translateX(-5000px) translateY(50px) translateZ(0px);
  }
  50% {
    -webkit-transform: translateX(0px) translateY(50px) translateZ(0px);
  }
  100% {
    -webkit-transform: translateX(5000px) translateY(50px) translateZ(0px);
  }
}

@-moz-keyframes moveFromLeftToRight {
  0% {
    -moz-transform: translateX(-5000px) translateY(50px);
  }
  50% {
    -moz-transform: translateX(0px) translateY(50px);
  }
  100% {
    -moz-transform: translateX(5000px) translateY(50px);
  }
}

@-ms-keyframes moveFromLeftToRight {
  0% {
    -ms-transform: translateX(-5000px) translateY(50px);
  }
  50% {
    -ms-transform: translateX(0px) translateY(50px);
  }
  100% {
    -ms-transform: translateX(5000px) translateY(50px);
  }
}

@-o-keyframes moveFromLeftToRight {
  0% {
    -o-transform: translateX(-5000px) translateY(50px);
  }
  50% {
    -o-transform: translateX(0px) translateY(50px);
```

```
    }
    100% {
      -o-transform: translateX(5000px) translateY(50px);
    }
  }

  @keyframes moveFromLeftToRight {
    0% {
      transform: translateX(-5000px) translateY(50px);
    }
    50% {
      transform: translateX(0px) translateY(50px);
    }
    100% {
      transform: translateX(5000px) translateY(50px);
    }
  }

  div.cloud {
    position: absolute;
    top: 0px;
    left: 0px;
    z-index: 500;
    background: transparent url(../../img/spritesheet.png) no-repeat
       -10px -257px;
    width: 566px;
    height: 243px;

    pointer-events: none;

    -webkit-animation-timing-function: linear;
    -webkit-animation-name: moveFromLeftToRight;
    -webkit-animation-duration: 2s;
    -webkit-animation-iteration-count: infinite;
    -webkit-animation-direction: alternate;

    -moz-animation-timing-function: linear;
    -moz-animation-name: moveFromLeftToRight;
    -moz-animation-duration: 60s;
    -moz-animation-iteration-count: infinite;
    -moz-animation-direction: alternate;

    -ms-animation-timing-function: linear;
    -ms-animation-name: moveFromLeftToRight;
    -ms-animation-duration: 60s;
    -ms-animation-iteration-count: infinite;
    -ms-animation-direction: alternate;

    -o-animation-timing-function: linear;
    -o-animation-name: moveFromLeftToRight;
    -o-animation-duration: 60s;
    -o-animation-iteration-count: infinite;
    -o-animation-direction: alternate;
```

```
    animation-timing-function: linear;
    animation-name: moveFromLeftToRight;
    animation-duration: 60s;
    animation-iteration-count: infinite;
    animation-direction: alternate;
  }
```

结果大致如图 5-5 所示。

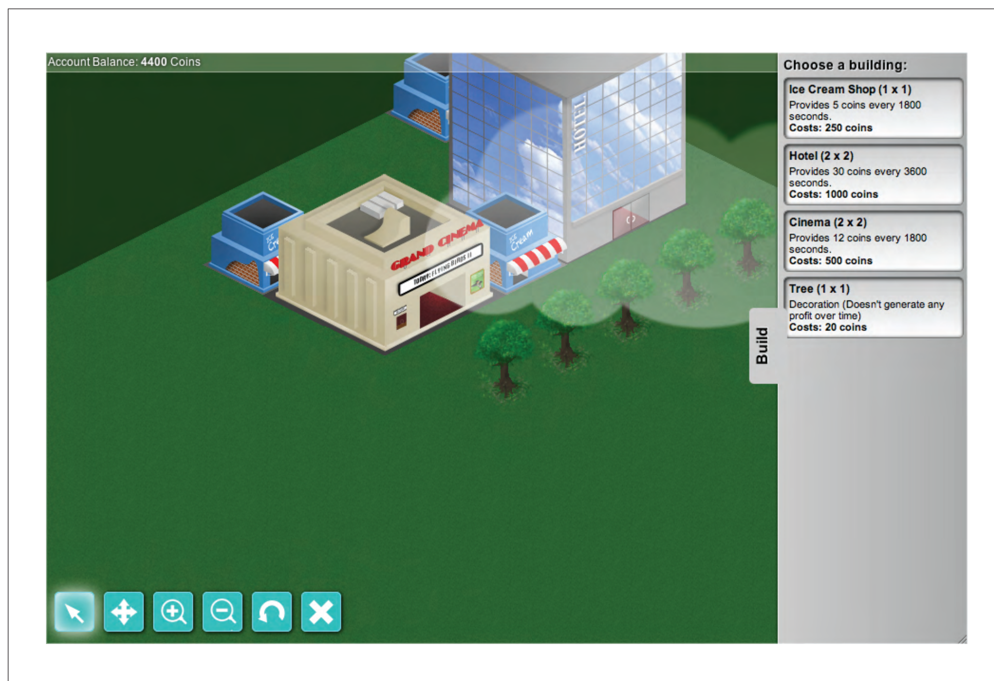


图 5-5　有云彩在城市上空飘过

另一个让场景更加可信的效果，就是为建筑物添加影子。在二维平面上创建投影的技术和算法有很多，但多数情况下，都需要一个或多个光源，也需要执行很多计算，特别是在有多个物体的情况下（我们的游戏中确实如此）。

因为场景中的对象大多数时候都是静止不动的，所以我们可以：

- 为每个建筑物都绘制一个阴影贴图；
- 我们自己动态地生成阴影。

在此，我们采取第二种方式，因为通过这种方式可以定义伪光源，也可以动态控制阴影的亮度。

这个方式分两步：首先，绘制原始建筑物的轮廓；其次，斜切并旋转轮廓并减少其 alpha 值。比如，假设光源来自右前方，那么阴影应该投向左后方。而如果光源来自左前方，那么阴影应该投向右后方。从图 5-6 可以看出我们想要实现的效果。

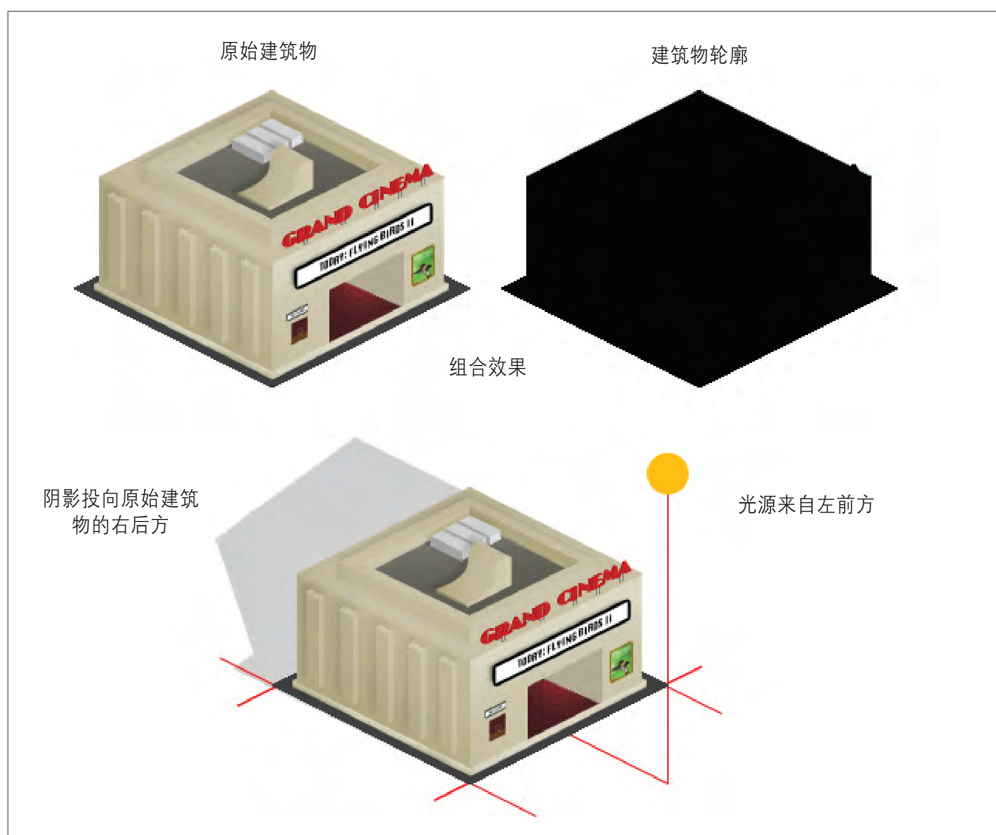


图 5-6　为建筑物添加阴影轮廓

为了实现这个效果，我们要利用之前介绍过的 HTML5 Canvas 的两个函数：getImageData() 和 putImageData()。而这个功能将直接写到 Sprite 对象中。

首先，修改 Sprite 类，添加一个空属性。

```
this.shadow = null;
```

接下来，修改 draw() 方法，让它接受一个可选的 drawShadow 参数。这个参数是一个布尔值，表示是否要为正在绘制的精灵绘制阴影。一个子例程会检查 this.shadow 是否为 null，必要时将图像转换成一个图像数组，以便将来调用 putImageData() 时使用。轮廓处理的结果将保存在 this.shadow 属性中，这样

下次再使用它的时候就不用再重复图像处理了。但不能不提醒读者，这个方法与使用预先制作的图像相比，效率确实要低很多，但我们可以有更好的控制和灵活性。

```
if (this.shown) {
  if (drawShadow !== undefined && drawShadow) {
    if (this.shadow === null) { // 还未创建阴影
      var sCnv = document.createElement("canvas");
      var sCtx = sCnv.getContext("2d");

      sCnv.width = this.width;
      sCnv.height = this.height;

      sCtx.drawImage(this.spritesheet,
        this.offsetX,
        this.offsetY,
        this.width,
        this.height,
        0,
        0,
        this.width * this.zoomLevel,
        this.height * this.zoomLevel);

      var idata = sCtx.getImageData(0, 0, sCnv.width, sCnv.height);

      for (var i = 0, len = idata.data.length; i < len; i += 4) {
        idata.data[i] = 0; // R
        idata.data[i + 1] = 0; // G
        idata.data[i + 2] = 0; // B
      }

      sCtx.clearRect(0, 0, sCnv.width, sCnv.height);
      sCtx.putImageData(idata, 0, 0);

      this.shadow = sCtx;
    }

    c.save();
    c.globalAlpha = 0.1;
    var sw = this.width * this.zoomLevel;
    var sh = this.height * this.zoomLevel;
    c.drawImage(this.shadow.canvas, this.posX, this.posY - sh, sw, sh * 2);
    c.restore();
  }

  c.drawImage(this.spritesheet,
     this.offsetX,
     this.offsetY,
     this.width,
     this.height,
     this.posX,
     this.posY,
     this.width * this.zoomLevel,
     this.height * this.zoomLevel);
}
```

最终结果如图 5-7 所示。

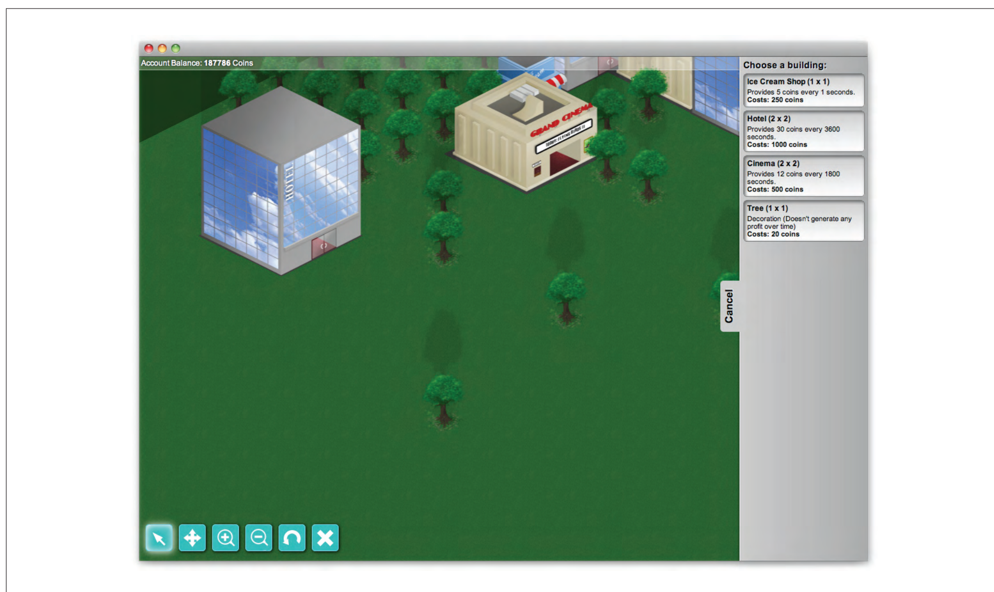


图 5-7　带影子的树

5.4　使用Facebook添加社交功能

使用 Facebook 添加社交功能只需要填几个表单、编少量代码即可实现。首先，需要在 http://www.facebook.com/developers/ 中单击“Create New App”按钮创建一个 Facebook 应用。单击之后，会出现 Facebook Application Creation 向导，引导我们把游戏集成到 Facebook 中。

单击“Create New App”按钮后，就会看到如图 5-8 所示的界面，要求你填写应用的名字。

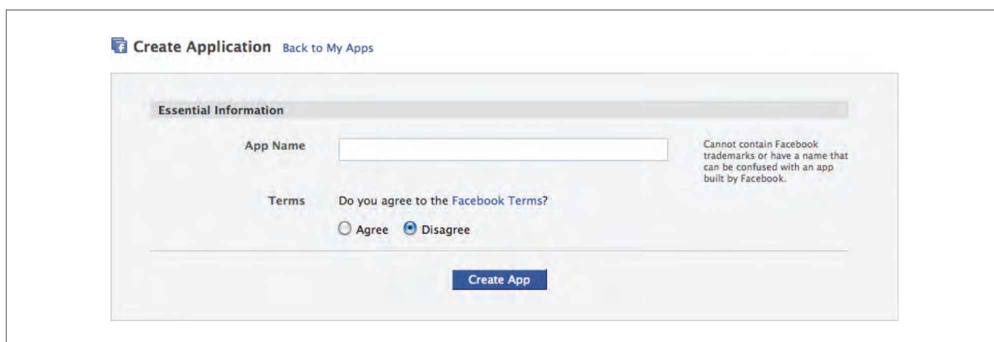


图 5-8　在 Facebook 中创建应用

填入你想要的名字（应该是游戏标题界面中显示的、用户在安装该应用时看到的名字），同意 Terms of Service 并单击“Create App”按钮。这里使用的名字是“Tourist Resort”。但应用的名字可以不必唯一，所以你也可以把自己的游戏命名为“Tourist Resort”。

下一步，如图 5-9 所示，是一个安全检查，按照提示把安全验证码填入文本框提交即可。



图 5-9　下一步：输入安全验证码

单击“Submit”按钮，取决你的账户信息及（或）所在位置，Facebook 可能会让你验证账户——通过关联的手机号码或者你名下的信用卡号码。

完成这些步骤后，就会来到应用管理界面。在这里，可以修改联系信息、应用名称、简介、标志、语言等细节信息。

我们可以选择是把应用集成到 Facebook 网页中，还是通过一个名为 Facebook Connect 的 Facebook 认证方案，将其作为一个单独的应用。关于 Facebook Connect 认证方案中所有选项，本书就不一一说明了，我们这里只介绍如何在 Facebook 的网页中显示游戏。

为了提供带有登录用户认证信息的外部站点，Facebook 使用了一个叫做 Canvas 的概念（跟 HTML5 Canvas 无关）。Facebook 的 Canvas 其实就是在一个 765 像素宽的 HTML `iframe` 中加载我们的站点，传入一些参数以便在服务器端或在客户端（使用 JavaScript 和 XBFML）完成 OAuth 2.0 认证。



本书只关注服务器端的实现。要了解 Facebook 平台支持的其他认证方案，请参考：http://developers.facebook.com/docs/authentication。

为了针对我们的游戏实现服务器端认证方案，需要单击应用管理页面左侧的“Facebook Integration”链接。第一个面板“Core Settings”如图 5-10 所示。

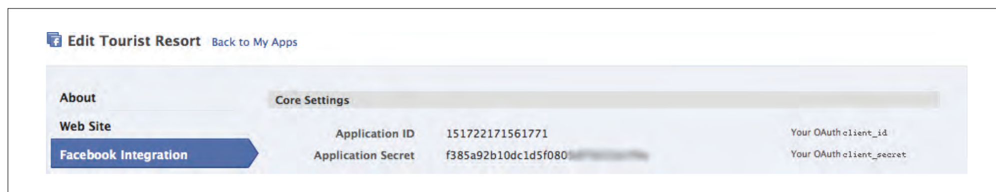


图 5-10　设置应用的关键信息

注意你的“Application ID”和“Application Secret”（图中遮盖了一部分密码），并下载 Facebook 官方的 PHP SDK：https://github.com/facebook/php-sdk/。

在“Facebook Integration”选项卡中，必须指定两个字段：

- Canvas Page，是一个基于 Facebook 的 URL，当用户单击我们的应用时就会打开这个 URL；
- Canvas URL，托管我们应用的 URL。

图 5-11 展示了这两个字段。

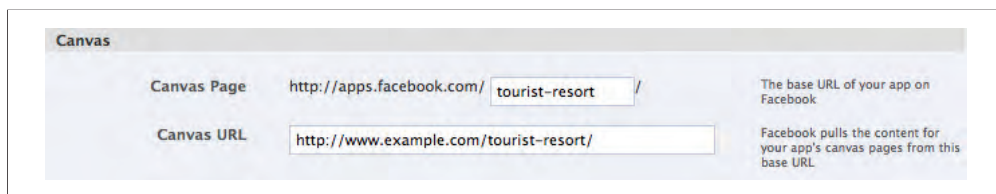


图 5-11　定义 Canvas

总而言之，当用户访问 http://apps.facebook.com/tourist-resort 时，Facebook 就会自动调用 Canvas URL 并传入参数以便完成验证。

然后，就是包含 Facebook API 文件，使用我们的“Application ID”和“Application Secret”创建一个 Facebook 对象的实例，就可以访问到用户信息了：

```
<?php
     require 'facebook/src/facebook.php';

     $fb = new Facebook(array(
           'appId' => '<YOUR_APPLICATION_ID>',
           'secret' => '<YOUR_APPLICATION_SECRET>',
      ));

     $user = $fb->getUser();
```

```
        echo '<pre>';
        print_r($user);
        echo '</pre>';
?>
```

如果用户没有登录到 Facebook 或者尚未对我们的应用授权，那么 `$user` 变量的输出将为 0。而检查这个情况有一个比较可靠的方法，就是使用 Facebook PHP SDK 中包含的 example.php 文件中的方式：

```
<?php
if ($user) {
    try {
        $user_profile = $fb->api('/me');
    } catch (FacebookApiException $e) {
        $user = null;
    }
}
?>
```

`$fb->api('/me')` 会查询 Facebook 的 Graph API（要了解有关 Facebook Graph API 的更多信息，请参考 http://developers.facebook.com/docs/reference/api/），获得已经向你授权的用户并将找到的用户赋值给 `$user_profile`，如果遇到错误则抛出异常。

为了让用户使用他们的 Facebook 凭据登录到我们的应用，还要添加如下代码：

```
<?php
$loginUrl = $fb->getLoginUrl();
?>
<a href="<?=$loginUrl?>">Sign In</a>
```

当用户单击“Sign In”后，Facebook 会显示请求授权对话框，如图 5-12 所示。

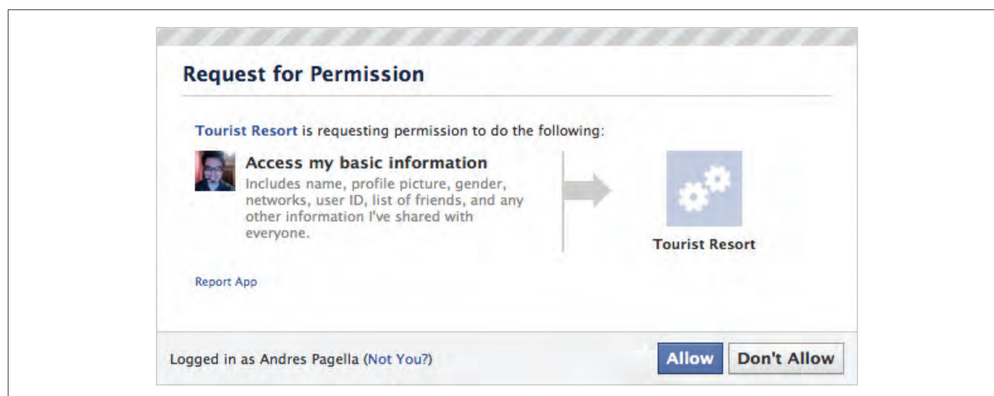


图 5-12　用户会不会授权我们访问他们的信息

在用户决定让我们的应用访问他们的数据之后，页面将会重新加载；这一次，`$user` 中将保存有 Facebook User Id，而 `$user_profile` 则会保存向 Graph API 查

询 /me 得到的响应，也就是一个 JSON 对象。

默认情况下，Facebook 只会请求“基本的用户信息”，包含用户名、姓、Facebook User ID、位置、兴趣、性别，等等。如果你还想得到用户授权，以便访问他们的电子邮件地址和朋友，甚至在他们的活动流中发表消息，那就需要把对 $fb->getLoginUrl() 的调用方式修改为如下所示：

```
<?php
$loginUrl = $fb->getLoginUrl(array(
    "scope" => "email, publish_stream, friends_about_me"
));
?>
```



要了解完整的授权项，请查看 http://developers.facebook.com/docs/authentication/permissions/。

修改了授权请求的代码之后，请求授权窗口将变成图 5-13 所示。

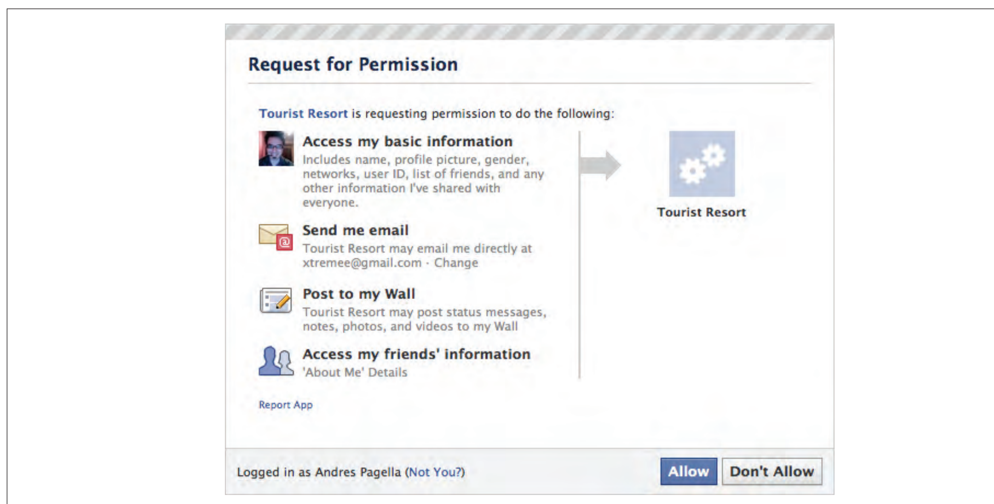


图 5-13　用户会不会授权我们访问他们更多的信息

如果用户同意我们的游戏访问他们的数据，那变量 $user_profile 中将包含一个电子邮件参数，可以用这个电子邮件在我们的数据库中自动注册用户。

在取得所有必要的信息后，只要在游戏中“一对一”地将用户 ID 映射到 Facebook User ID，让他们在通过 Facebook 连接到游戏时自动登录即可。

如果要了解相关的更多信息，请参考 Facebook Developer 站点：http://developers.facebook.com/docs/guides/canvas/。

深入HTML5应用开发

本书合并了O'Reilly出版的《HTML5 Geolocation》与《HTML5：等轴实时游戏开发》两本书的内容。

第一部分介绍了HTML5 Geolocation API。使用这种API，开发人员不必针对特定设备编程，就能够在浏览器中直接编写地理定位应用。这部分共6章，介绍了Geolocation API在浏览器代码中的使用，并通过大量示例向读者展示其"一次编写，随处部署"的特点。具体内容包括地理定位的基础知识简介，这套API的浏览器支持情况，以及如何利用它和其他常用地图工具在网页中实现类似Google地图的嵌入式地图。

这一部分的主要内容还有：

- 根据设备的不同，从各种来源收集地理信息
- 探索地理坐标系统，包括大地测量系统和基准点
- 使用Geolocation API，以JavaScript代码从用户的浏览器中取得位置信息
- 使用Google地图或基于JavaScript的ArcGIS API将位置信息显示在地图上
- 使用数据库、KML文件和Shapefile保存地理信息
- 熟悉地理数据的实际用途，包括地理营销、地理社交、地理标签和地理应用

第二部分介绍了用HTML5、CSS3和JavaScript开发等轴实时游戏。任何沉迷过Zynga的《开心农场》的玩家都享受过这类游戏的乐趣，本书展示了如何设计并开发这类游戏，涵盖了只使用开源工具来实现项目的全过程。你将通过详细的示例和代码，学会绘制图形、使用精灵、添加声音和验证数据以防作弊。

我们最后会用这部分介绍的所有技术完成一个《旅游胜地》游戏，并将它连接到社交网络上。如果你会用HTML5、CSS3和JavaScript，那就可以准备开始了！

这一部分的主要内容如下：

- 使用HTML5的画布（canvas）元素和精灵创建流畅的动画
- 创建高性能的等轴网格区块
- 设计同时适用于桌面设备和移动设备的游戏界面
- 使用HTML的音频（audio）元素为游戏添加声音
- 用Web Workers实现游戏中的路径查找功能
- 用PHP和MySQL实现客户端数据模型
- 使用动态CSS3对象让游戏更有活力

封面设计：Karen Montgomery　张健

图灵社区：www.ituring.com.cn
反馈/投稿/推荐信箱：contact@turingbook.com
热线：(010)51095186转604

分类建议　计算机 / Web开发

人民邮电出版社网址：www.ptpress.com.cn

O'REILLY®
oreilly.com.cn

O'Reilly Media, Inc.授权人民邮电出版社出版



ISBN 978-7-115-27494-6
定价：49.00元

欢迎加入

图灵社区

最前沿的IT类电子书发售平台

电子出版的时代已经来临。在许多出版界同行还在犹豫彷徨的时候，图灵社区已经采取实际行动拥抱这个出版业巨变。作为国内第一家发售电子图书的IT类出版商，图灵社区目前为读者提供两种DRM-free的阅读体验：在线阅读和PDF。

相比纸质书，电子书具有许多明显的优势。它不仅发布快，更新容易，而且尽可能采用了彩色图片（即使有的书纸质版是黑白印刷的）。读者还可以方便地进行搜索、剪贴、复制和打印。

图灵社区进一步把传统出版流程与电子书出版业务紧密结合，目前已实现作译者网上交稿、编辑网上审稿、按章发布的电子出版模式。这种新的出版模式，我们称之为“敏捷出版”，它可以让读者以较快的速度了解到国外最新技术图书的内容，弥补以往翻译版技术书“出版即过时”的缺憾。同时，敏捷出版使得作、译、编、读的交流更为方便，可以提前消灭书稿中的错误，最大程度地保证图书出版的质量。

最方便的开放出版平台

图灵社区向读者开放在线写作功能，协助你实现自出版和开源出版的梦想。利用“合集”功能，你就能联合二三好友共同创作一部技术参考书，以免费或收费的形式提供给读者。（收费形式须经过图灵社区立项评审。）这极大地降低了出版的门槛。只要你有写作的意愿，图灵社区就能帮助你实现这个梦想。成熟的书稿，有机会入选出版计划，同时出版纸质书。

图灵社区引进出版的外文图书，都将在立项后马上在社区公布。如果你有意翻译哪本图书，欢迎你来社区申请。只要你通过试译的考验，即可签约成为图灵的译者。当然，要想成功地完成一本书的翻译工作，是需要有坚强的毅力的。

最直接的读者交流平台

在图灵社区，你可以十分方便地写作文章、提交勘误、发表评论，以各种方式与作译者、编辑人员和其他读者进行交流互动。提交勘误还能够获赠社区银子。

你可以积极参与社区经常开展的访谈、审读、评选等多种活动，赢取积分和银子，积累个人声望。

ituring.com.cn